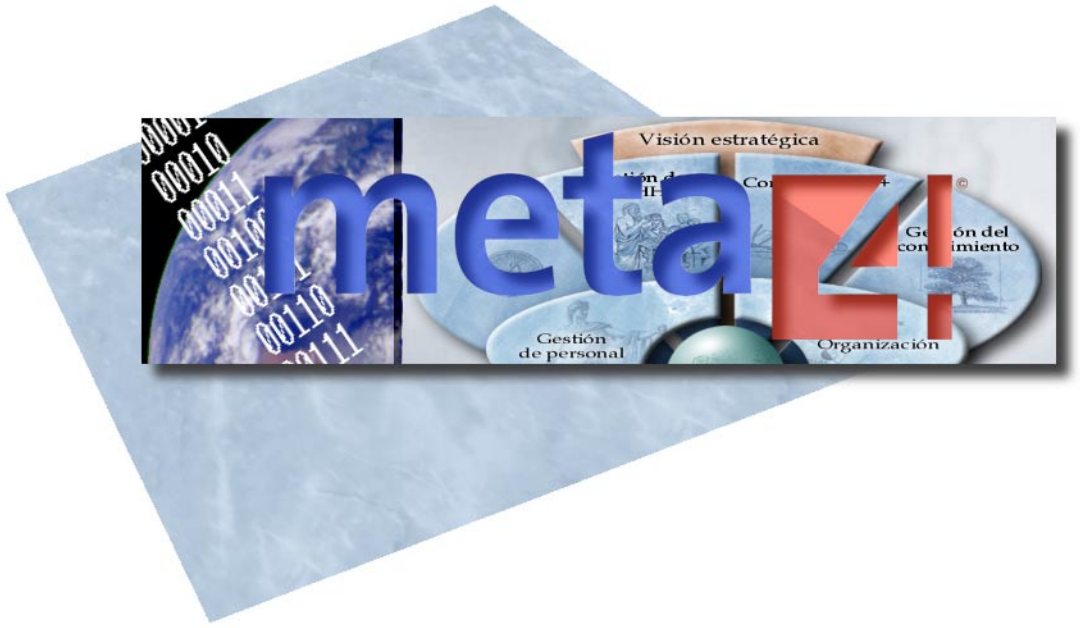
Tecnología Meta4: Manual de referencia

Lenguaje LN4



Meta4 Spain, S.A.
Centro Europa Empresarial
Edificio Roma
C/ Rozabella, 8
Ctra. de La Coruña, km 24,200
28230 Las Rozas, Madrid
ESPAÑA
http://www.meta4.com

Fecha de creación: 20 de enero de 2003.

Fecha de la última publicación: 30 de enero de 2006

Tabla de contenidos

Manual de referencia del lenguaje LN4

Introducción

El lenguaje de programación LN4 es un potente y versátil lenguaje de desarrollo, exclusivo de aplicaciones de Meta4 como Meta4 PeopleNet y Meta4 KnowNet, que ofrece diversas posibilidades al desarrollador.

Entre sus características más destacables, se pueden citar:

- Asignación dinámica de tipos de datos, sin declarar las variables de manera explícita.
- Estructuras de control propias de cualquier lenguaje de programación, tales como:
  - Ejecución condicional: If...then...else
  - Sentencias repetitivas:
    - While...wend
    - For...next
  - etc.
- Facilidades para el diseño de fórmulas que calculen el valor de los conceptos, o realicen cualquier tipo de cálculo sobre datos.
- Utilización de elementos de las estructuras de nodo de cualquier Meta4Object en los cálculos y procesos, lo que permite referenciar valores asignados a otros conceptos o campos en el resto de estructuras de nodo (EN) de la aplicación.

El objetivo del presente manual consiste en describir todos los elementos del lenguaje, así como la forma de usar éstos para sacar el mayor provecho de los mismos.

Lenguaje LN4 y estructuras de nodo

Aunque los elementos de las estructuras de nodo pueden ser también de tipo **campo** (contienen valores extraídos directamente de la base de datos) o de tipo **propiedad** (contienen únicamente valores que se asignen a cada una de las propiedades), el código LN4 puede estar asociado únicamente a los elementos de tipo concepto o de tipo método.

Los **conceptos** adquieren un valor que se genera a partir de la fórmula LN4 a la que estén asociados o aquel valor que se le asigne como si de una propiedad se tratase.

Por su parte, los **métodos** equivalen a las funciones y procedimientos de los lenguajes de programación. En este caso, ejecutan una serie de instrucciones LN4 y pueden utilizarse entre otras acciones para actualizar los elementos de tipo campo de aquellas estructuras de nodo que pertenecen al mismo Meta4Object u otro diferente.

La ejecución de los conceptos o métodos puede realizarse a nivel de:

- **Nodo**

  En este caso, hay que estar posicionado en el nodo específico sobre el que se va a actuar, el cual debe estar activo.

- **Bloque**

  Este debe estar activo, ya que en principio no debería afectar al resto de bloques.

- **Registro**

  Este debe estar activo, ya que en principio no debería afectar al resto de los registros.

Llamadas a los valores de los elementos de las estructuras de nodo

Desde las fórmulas creadas con LN4 se pueden utilizar:

- Los valores asignados a los elementos de la estructura de nodo en la que se define el método o concepto.
- Los valores asignados a los elementos de otras estructuras de nodos que estén definidas en el mismo Meta4Object que la estructura de nodo en la que se define el método o concepto.
- Los elementos de tipo método de la misma estructura de nodo u otras estructuras del mismo Meta4Object.

- Los elementos de tipo método de la misma estructura de nodo u otras estructuras de distintos Meta4Objects.

Estas asignaciones se graban físicamente en la base de datos cuando se ejecuta la función Persist_tree ( ). Esto sucede únicamente en el caso de los campos, pero no en el resto de elementos.

Para llevar a cabo las llamadas a los elementos de la estructura de nodo, se utilizará la siguiente sintaxis:

- Para referenciar un elemento de la estructura de nodo en la que se está creando el concepto o método, basta con escribir el nombre del elemento. El intérprete de LN4 toma el valor que se haya asignado al elemento en el registro que se encuentre activo.

  Si se quiere referenciar un elemento de la estructura de nodo en un registro distinto del registro activo, se debe cambiar el registro activo mediante la función MoveTo (ÍndiceRegistro).

- Para referenciar un elemento de otro nodo que forme parte del mismo Meta4Object, basta con escribir el alias de la estructura de nodo asociada al nodo, seguido de un punto y del nombre del elemento que se quiere utilizar.

  ```
  AliasEN.NombreItem
        o
  EN.NombreItem
  ```

  El alias debe ser un identificador que distinga de forma unívoca a cada nodo dependiente de una misma estructura de nodo.

- Para referenciar un elemento de un registro específico en un nodo incluido en el mismo Meta4Object, es necesario indicar detrás del nombre de la estructura de nodo, entre corchetes, el índice del registro cuyo elemento se quiere utilizar. A continuación se debe escribir un punto y el nombre del elemento.

  ```
  NombreEN[índice].NombreItem
        ó
  AliasEN[índice].Item
  ```

- Para referenciar un elemento con tramos delimitados por una fecha de inicio y una fecha de fin:
  - Si se trata de un elemento de la estructura de nodos sobre el que se está trabajando, es suficiente con escribir el nombre del elemento y, entre corchetes, el número del tramo o una fecha del rango al que se desea hacer referencia.

Los tramos se numeran comenzando por 0 para el primer tramo, el 1 para el segundo, y así sucesivamente. De esta forma, por ejemplo, la asignación de un valor al tramo del elemento Salario_Base situado en el registro activo comprendido entre las fechas 01-03-1999 y 15-03-1999, se podría hacer de la siguiente forma:

```
Salario_Base [ {1999-03-10} ] = 120000
```

Si se conoce el lugar que ocupa el tramo, sería posible utilizar su índice en lugar de una fecha. Así pues, si el tramo comprendido entre el 01-03-1999 y el 15-03-1999 fuese el segundo tramo, se le podría asignar un valor mediante la correspondiente instrucción:

```
Salario_Base [ 1 ] = 120000
```

- Para asignar valor a este mismo elemento y tramo, en el registro situado en séptimo lugar de la estructura de nodo llamada Acumulado_Largo, se debería utilizar la instrucción:

```
Acumulado_Largo [ 6 ].Salario_Base [ {1999-03-10} ]
```

Se debe emplear esta misma sintaxis para recuperar el valor almacenado en un elemento con tramos y asignarlo a variables.

Por ejemplo, la siguiente instrucción asignaría a la variable SalBase el valor del segundo tramo del elemento Salario_Base del quinto registro del nodo Acumulado_Largo.

```
SalBase = Acumulado_Largo[4].Salario_Base[1]
```

Elementos básicos del lenguaje

Variables y constantes

El nombre de las variables debe comenzar por una letra o el carácter de subrayado, seguido por un conjunto de letras y/o números. En ninguno de ambos casos es posible incluir espacios en blanco o caracteres especiales.

Para nombrar a las variables, es recomendable utilizar identificadores fáciles de recordar, con la condición de que no excedan los 255 caracteres.No obstante, por usabilidad se recomienda un máximo de 100 caracteres.

Las variables LN4 no necesitan ser declaradas de manera explícita, sino que basta con inicializarlas asignándoles un valor. Asimismo, tampoco están tipificadas, es decir, el tipo de dato de una variable cambia dependiendo de los valores que se le asignen.

La consecuencia inmediata es que si se intenta leer el contenido de cualquiera de estas variables antes de haberles asignado un valor de tipo fecha, cadena o número, se produce un error.

Tipos de datos

Una variable puede almacenar un valor dependiendo del tipo de dato con el que haya sido inicializada. Este tipo puede ser uno de los siguientes:

- Fecha
- Número
- Cadena

Fecha

Este tipo de variable aparece en formato TS (Iso TimeStamp) y entre llaves. No es necesario especificar todo el conjunto de datos.

EJEMPLO

```
{1997-12-01 12:59:59}
```

Número

Este tipo de variable permite utilizar los formatos decimal (con un punto para separar la parte entera de la decimal), de notación científica, y hexadecimal (éste último sólo para números enteros).

La notación científica permite expresar cantidades muy grandes de forma sencilla, anotando los números como potencia de base diez, lo que facilita la realización de operaciones matemáticas en problemas científicos.

En el sistema decimal, cualquier número real puede expresarse mediante la denominada notación científica normalizada. Para expresar un número en notación científica normalizada, se multiplica o se divide por 10 tantas veces como sea necesario para que todos los dígitos aparezcan a la derecha del punto decimal y que el primer dígito después del punto no sea cero.

EJEMPLO

$732.5051 = 0.7325051 \times 10^{3}$

$-0.005612 = -0.5612 \times 10^{-2}$

Cadena

El texto que aparezca en esta variable debe ir entre comillas.

Llamadas a funciones

Las llamadas a una función cualquiera se realizan de forma similar a otros lenguajes de programación. Dependiendo de la función, se deberán o no indicar argumentos en el momento de realizar la llamada.

```
Opción 1: [Variable =] nombre_función (argumentos)
Opción 2: nombre_función ( )
```

Retorno de valores

Para obtener el valor de una variable, se utiliza la siguiente sintaxis:

```
return (expresión)
```

Operadores LN4

El lenguaje LN4 recoge un conjunto de operadores que permiten realizar operaciones de diverso tipo sobre variables. Cada uno de estos operadores pertenece a una de las siguientes categorías:

- Operadores matemáticos binarios
- Operadores matemáticos unarios
- Operadores lógicos
- Operadores de comparación

La siguiente tabla recoge los distintos operadores que pertenecen a cada categoría:.

Operadores matemáticos binarios	+	Suma
	-	Resta
	*	Multiplicación
	/	División
Operadores matemáticos unarios	-	Indicador de número negativo
Operadores lógicos	NOT	Devuelve VERDADERO si es cierta la condición situada a la izquierda del operador y falsa la condición situada a la derecha del operador.
	AND	Devuelve VERDADERO si son ciertas las dos condiciones situadas a la izquierda y a la derecha del operador.
	OR	Devuelve VERDADERO si es cierta al menos una de las condiciones situadas a la izquierda o a la derecha del operador.

Operadores de comparación	=	Igual que
	<>	Distinto de
	>	Mayor que
	<	Menor que
	>=	Mayor o igual que
	<=	Menor o igual que

Comentarios en las fórmulas

El lenguaje LN4 permite introducir comentarios en las fórmulas. Estos deben ir precedidos por unos caracteres especiales que indican al compilador que debe hacer caso omiso del texto comprendido entre ellos.

A la hora de incluir un comentario en el código de LN4, se pueden dar dos situaciones:

- Si el comentario no ocupa más de una línea, debe ir precedido por el carácter ‘, o por los caracteres //.
- Si el comentario ocupa más de una línea, debe ir precedido por los caracteres /*, y detrás de la última palabra del comentario se deben añadir los caracteres */.

Expresiones regulares

Las expresiones regulares son expresiones que permiten establecer criterios de selección o búsquedas sobre el conjunto de elementos con los que se trabaja.

Con LN4 existe la posibilidad de trabajar con expresiones regulares, tanto para permitir búsquedas potentes con la función FindRegister ( ), como para buscar expresiones regulares dentro de un texto cualquiera con la función MatchRegister ( ).

La sintaxis de estas expresiones regulares es la siguiente:

Expresión	Explicación de la expresión utilizada
( )	Este operador permite agrupar parte de una expresión regular y que se repita una o más veces cierta subexpresión regular. Así por ejemplo, la expresión **a\(bc\)+** encontrará una **a** seguida de uno o más bloques de **bc**.
|	Este operador permite establecer un OR entre dos subexpresiones. Por ejemplo, **ab|ac** recuperará las secuencias **ab** y **ac**.
+, *	El operador * indica que la expresión que le precede puede aparecer cero o más veces y el operador + indica que aparece una o más. Por ejemplo, **ab*** encontrará una **a** seguida de ninguna o varias **b**.
?	Este operador indica que la expresión que le precede es opcional. Por ejemplo, **p?sico** encontrará **psico** y **sico**.
$ y ^	El operador $ indica que la expresión que le precede debe ser el final del texto y el operador ^indica que la expresión que le sigue debe ser la primera del texto. Por ejemplo, **^en** encontrará los textos que empiecen por en, no otros. Por ejemplo, **adiós$** sólo encontrará los textos que acaban con la palabra adiós.
.	El operador **.** es un comodín que permite encontrar cualquier carácter. Por ejemplo, **a.*b** encontrará las cadenas que empiecen por **a** y acaben en **b**.
\w, \W	El operador **\w** encuentra cualquier carácter que pueda formar parte de una palabra, es decir, letras mayúsculas/minúsculas y números. El operador **\W** encuentra los que no pueden formar parte de una palabra. **A\wB** encontrará **acb**, pero no **a b**
\< \>	Si \< precede a una subexpresión, indica que la subexpresión debe encontrarse al principio de una palabra. Si \> sucede a una subexpresión, sólo encontrará las palabras que acaben con esa expresión. Por ejemplo, **\<mega** sólo encuentra las palabras que empiecen por mega. Por ejemplo, **mente\>** sólo recuperará las palabras que acaben en mente.
**Caracter **	Este operador \\ permite encontrar el carácter de escape.

Estructuras de control

El lenguaje LN4 incluye diversas estructuras de control que permiten, entre otras funciones, establecer ejecuciones cíclicas de acciones, seleccionar elementos que cumplan ciertas condiciones, etc. Estas estructuras de control, cuya sintaxis y modo de uso veremos a continuación, se dividen en:

- Sentencias condicionales
  - *If… Then*, *If… Then… Else*, etc.
- Bucles de repetición de sentencias tales como
  - *Do… Until*
  - *While… Wend*
  - *For… Next*

Dentro de estas estructuras de control existe la posibilidad de indicar dos acciones en la misma línea, separándolas únicamente por ":".

EJEMPLO

```
If A < 10 then
   a=12 : b=16
Else
   Messagebox ( "Aviso", "El valor de A es igual o mayor que 10 ")
EndIf
```

Sentencias condicionales

Sentencia If … Then … EndIf

EJEMPLO

```
If condición Then
...
Endif
```

La sentencia *If … Then … EndIf* se utiliza para ejecutar una serie de sentencias cuando se cumplan una o varias condiciones, que se establecen como criterios de selección.

El orden de esta sentencia es el siguiente:

- Las condiciones se escriben inmediatamente después de la palabra reservada *If*
- Las sentencias que se han de ejecutar si se cumplen las condiciones, deben escribirse entre las palabras reservadas *Then* y *EndIf.*

Es posible anidar tantas sentencias *If… Then… EndIf* como sea necesario.

- La palabra reservada *EndIf* debe escribirse sin interponer un espacio en blanco entre *End* e *If*, ya que de hacerse así se produciría un error.
- En caso de que se indiquen dos o más condiciones detrás de la palabra reservada *If*, éstas deben unirse mediante los operadores lógicos AND, OR y/o NOT:
  - Si se unen varias condiciones mediante el operador **AND**, las sentencias indicadas entre las palabras reservadas *Then* y *EndIf* se ejecutarán sólo si se cumplen todas las condiciones.

EJEMPLO

Sea el siguiente fragmento de código LN4:

```
If  A < 10 AND A > 5 Then
   MessageBox ( "Aviso", " El valor de A es " , A )
EndIf
```

En este caso, si la variable A recoge un valor comprendido entre 5 y 10, se mostrará el valor de la variable en un cuadro de diálogo. Sin embargo, si el valor de A no está comprendido en este intervalo, no se ejecutará ninguna acción.

  - Si se indica como condición precedida por el operador **NOT**, las sentencias indicadas entre las palabras reservadas *Then* y *Endif* se ejecutarán si no se cumplen las condiciones indicadas.
  - Si se unen dos condiciones mediante el operador lógico **OR**, las sentencias indicadas entre las palabras reservadas *Then* y *EndIf* se ejecutarán si se cumple al menos una de las condiciones.

EJEMPLO

Sea el siguiente fragmento de código LN4:

```
If  A < 10 OR B > 5 Then
   MessageBox ( "Aviso", " El valor de A es " , A )
EndIf
```

En este caso, si la variable A contiene un valor superior a 10, o la variable B contiene un valor superior a 5, entonces se mostrará el valor de la variable A en un cuadro de diálogo.

Si no se cumple ninguna de estas dos condiciones, no se ejecutará ninguna acción.

Sentencia If … Then … Else … EndIf

EJEMPLO

```
If condición then
...
Else
...
EndIf
```

Esta estructura se utiliza para seleccionar el conjunto de sentencias que se ejecutarán, en función del valor de la condición o condiciones indicadas tras la palabra reservada *If*.

Es posible indicar más de una condición, uniéndolas con los operadores lógicos AND y OR.

- Si se cumplen las condiciones, se ejecutarán las sentencias que se hayan escrito entre las palabras reservadas *Then* y *Else*.
- Si no se cumple la condición o condiciones indicadas, se ejecutarán las sentencias que se hayan escrito entre las palabras reservadas *Else* y *EndIf*.

Si la estructura ***If...Then...Else*** se ha escrito en una sola línea, no es necesario finalizar con la palabra ***Endif***.

EJEMPLO

El código de este ejemplo realiza lo siguiente:

- Si la variable A recoge un valor situado en el rango delimitado por los números 5 y 10, aparece un mensaje en pantalla con el valor de la variable.
- Si el valor de A no está comprendido en este intervalo, se le suma 10 al valor de la variable y se mostrará un mensaje en pantalla con el valor obtenido tras realizar esta operación.

```
If A > 5 AND A < 10 Then
   Messagebox ( "Aviso", "El valor de A es ", A )
Else
   A = A + 10
   Messagebox ( "Aviso", "El valor de A tras sumarle 10 es ", A )
EndIf
```

Sentencia If …Then …ElseIf …Else …EndIf

EJEMPLO

```
If condición then
...
Elseif condición then
...
Else
...
Endif
```

Esta estructura de control se utiliza para realizar una serie de acciones en función de las condiciones indicadas. Así, si se cumple la condición o condiciones indicadas tras la palabra reservada *If*, se ejecutarán únicamente las sentencias escritas entre la palabra reservada *Then* y el primer *ElseIf*.

Si no se cumple la primera condición o condiciones, se evaluarán las condiciones que se indican detrás de la primera palabra reservada *ElseIf*. Si se cumplen éstas, se ejecutará el código que se haya escrito entre las palabras reservadas *ElseIf* y *Else*. Si no se cumple, se ejecutará el código escrito entre las palabras reservadas *Else* y *EndIf*.

Dentro de esta estructura es posible incluir tantas cláusulas *ElseIf* como se consideren necesarias. También es posible omitir la cláusula *Else*, para que así

no se ejecute ninguna opción en el caso de que no se cumplan las condiciones indicadas en las claúsulas *If* y *ElseIf*.

EJEMPLO

El código del siguiente ejemplo realiza las siguientes acciones.

- Si A recoge un valor comprendido entre 5 y 10, se muestra en pantalla un cuadro de diálogo con su valor.
- Si el valor de A es mayor que 10, se eleva su valor al cuadrado y se muestra en pantalla el resultado. En cualquier otro caso, es decir, si A recoge un valor menor que 5, se le suma 15.
- En todos los casos se mostrará un cuadro de diálogo que indica que las operaciones han finalizado.

```
If A > 5 AND A < 10 Then
   MessageBox ( "Aviso", "El valor de A es ",  A )
ElseIf A > 10 Then
   A = Power ( A, 2 )
   MessageBox ( "Aviso", "El valor de A al cuadrado es ", A  )
Else
   A = A + 15
EndIf
MessageBox ( "Aviso", "Las operaciones han finalizado" )
```

Estas sentencias también pueden escribirse anidando sentencias If … Then, de la siguiente forma:

```
If A > 5 Then
   If A < 10 Then
      MessageBox ( "Aviso", "El valor de A es ", A )
   Else
      A = Power ( A, 2 )
      MessageBox ( "Aviso", "El valor de A al cuadrado es ",  A )
   EndIf
Else
   A = A + 15
EndIf
MessageBox ( "Aviso", "Las operaciones han finalizado" )
```

Sentencias de repetición

Sentencia Do … Until condición

EJEMPLO

```
Do
...
Until condición
```

La sentencia *Do … Until* ejecuta el conjunto de sentencias situado entre las palabras reservadas *Do* y *Until*, hasta que se cumpla la condición o condiciones indicadas tras la palabra reservada *Until*.

En el caso de que se indiquen dos o más condiciones, éstas deben unirse mediante los operadores lógicos AND y/o OR.

EJEMPLO

El siguiente código recorre los campos de una EN hasta alcanzar la variable lógica EOF. Mientras no se encuentre, se suma el valor del campo SALARIO_BASE de todos los registros de la EN al campo TOTAL_SALARIO de la misma estructura de nodos.

```
TotalSalario = 0
Begin ( )
Do
   TotalSalario = TotalSalario + SALARIO_BASE
   Gonext ( )
Until IsEOF ( ) = M4_TRUE
MessageBox ("Aviso", "El total de los salarios es ", TotalSalario )
```

Sentencia While condición … Wend

EJEMPLO

```
While condición
...
Wend
```

La sentencia *While … Wend* se utiliza siempre que se tenga que repetir una serie de instrucciones, mientras se cumpla una o varias condiciones establecidas previamente.

- Si se unen dos condiciones mediante el operador AND, las instrucciones se ejecutarán mientras se cumplan las dos condiciones.
- Si se unen dos condiciones mediante el operador OR, las instrucciones se ejecutarán mientras se cumpla al menos una de las condiciones.

EJEMPLO

El siguiente código recorre los registros de la estructura de nodos asociada al nodo activo, hasta que se encuentre un registro cuyo campo SALARIO_BASE sea mayor que 600.000, o alcance la variable lógica EOF.

Para cada registro se suma el valor del campo SALARIO_BASE en la variable TotalSalario. Finalmente, se muestra el resultado en un cuadro de diálogo.

```
SalarioBase = 0
Begin ( )
While Not IsEOF ( )=M4_TRUE AND SALARIO_BASE < 600000
   SalarioBase = SalarioBase + SALARIO_BASE
   Gonext ( )
Wend
MessageBox ("Aviso", "El total del salario base es ", SalarioBase )
```

Sentencia For … Next

EJEMPLO

```
For variable = n to m
...
Next
```

La estructura *For … Next* repite una o varias instrucciones el número de veces que se indique detrás de la palabra reservada *For*.

Cada vez que se repite el conjunto de instrucciones, se incrementa un contador interno, cuyo valor se guarda en la variable de control. Este contador puede utilizarse, por ejemplo, para recorrer los registros de una estructura de nodos.

Si se hace así, recuerde que los registros de una EN se numeran comenzando por el número 0. No es posible modificar el incremento del contador en la línea del For; si se necesita incrementar este valor con un número distinto a 1, se debe hacer aumentando la variable con el valor deseado en el cuerpo del bucle.

La sentencia For...Next sólo permite incrementar la variable indicada de uno en uno, aunque es posible incrementar el valor de la variable utilizada dentro de la sentencia For...Next un número mayor.

EJEMPLO

El siguiente código muestra seis veces un cuadro de diálogo. El cuadro de diálogo indica en pantalla el valor que recoge la variable n en cada momento.

```
For n = 1 To 6
   MessageBox ( "Aviso", "El contador n recoge el valor ", n )
Next
```

Manejo de elementos

Elementos del sistema

Un elemento del sistema es cualquier elemento que pertenezca a una estructura de nodos determinada. El fin de estos elementos es disponer de un grupo de ellos que mapeen directamente un conjunto de funciones de C++ para ejecutar en el Meta4Object Engine.

Este conjunto de elementos proporcionan funcionalidades básicas, tales como, búsquedas, navegación, ejecución, etc.

Sintaxis de lectura y asignación de elementos

A continuación se detalla la sintaxis necesaria para realizar una asignación de valor a los elementos y proceder posteriormente a su lectura:

- elemento
- elemento[tramo]
- nodo.elemento
- nodo[registro].elemento
- nodo[registro].elemento[tramo]

Por ejemplo, la asignación de valores a un elemento correspondiente a un nodo se haría del siguiente modo:

```
nodo.elemento = valor
```

Por último, para hacer referencia a un Meta4Object distinto al actual, se debe utilizar la siguiente sintaxis:

```
Alias ! .
```

Atributos y Atributos-Métodos

En el apartado Sintaxis de lectura y asignación de elementos se indica cómo asignar valores a elementos con tramos y cómo recuperar los valores que contienen los distintos tramos de un elemento.

Las propiedades y métodos de ambos se indican en la tabla siguiente:

Propiedades	Aplicable a	Tarea
SysSliceNumber	Elemento	Devuelve el número de tramos que contiene el elemento.
SysStartDate	Tramo	Fecha de inicio del tramo.
SysEndDate	Tramo	Fecha de fin del tramo.
SysFirstSlice	Elemento	Índice del primer tramo del elemento.
SysLastSlice	Elemento	Índice del último tramo del elemento.
SysOldValue	Elemento y tramos	Devuelve el valor antiguo del elemento cuando se le haya cambiado el valor desde LN4.
SysAddNewSlice (fecha1, fecha2)	Elemento	Añade un nuevo tramo al elemento. Recibe como argumentos la fecha de inicio y de fin del tramo.
SysGetItemName ()	Elemento	Devuelve el nombre del elemento.
SysBlobDescription	Elemento	Descripción blob del elemento.
SysBlobMask	Elemento	Máscara blob del elemento (entero).
SysAuxiliarItemID	Elemento	ID Elemento auxiliar (cadena)
SysAddNewSlice (start_date, end_date)	Elemento y tramos	Añade un nuevo tramo al elemento.
SysGetItemName (Nombre)	Elemento	Devuelve el valor del elemento en un argumento (por referencia)
SysCreateSlice (start_date, end_date, value)	Elemento y tramos	Añade un nuevo tramo al elemento y le asigna un valor.
SysSplitSlice (date)	Tramos	Divide el tramo en dos por esta fecha, manteniendo el valor acorde al comportamiento.
SysTotalItems (function)	Elemento	Totaliza los elementos.
SysTotalslices (function)	Elemento	Totaliza los tramos.
SysCreateBlobFile (Directory, extension)	Elemento	Crea un archivo blob para el elemento en el directorio con esta extensión.
SysSetBlobDescription (description)	Elemento	Asigna la descripción blob de este elemento.

Para emplear estas propiedades y métodos, debe escribirse el nombre del elemento seguido del identificador del tramo (cuando la propiedad sea aplicable a tramos), seguido de dos puntos (..), y el nombre de la propiedad o método que se quiere emplear.

EJEMPLO

Item..SysSliceNumber	Devuelve el número de tramos del elemento
Item[Tramo]..SysStartDate	Devuelve la fecha de inicio del tramo que se indica. Si no se indica ningún tramo, devuelve la fecha de inicio del tramo actual.
Item[Tramo]..SysEndDate	Devuelve la fecha de fin del tramo que se indica. Si no se indica ningún tramo, devuelve la fecha de fin del tramo activo.
Item..SysFirstSlice	Devuelve el índice del primer tramo del elemento. Siempre devolverá 0, incluso si el elemento no contiene ningún tramo.
Item..SysLastSlice	Devuelve el índice del último tramo del elemento. Por ejemplo, si el elemento tiene 6 tramos, esta propiedad recogerá el valor 5. Si el elemento tuviese un tramo, devolverá el valor 0. Si el elemento no tiene tramos, devolverá 0.
Item..SysAddSlice ({1999-07-01},{1999-07-31})	Crea un nuevo tramo en un elemento con fecha de inicio 01-07-1999 y fecha de fin 31-07-1999.
Item = 34000 ‘Cambiamos el valor de elemento intValorAnterior = Item..SysOldValue	Devuelve el valor que tenía el elemento antes de asignarle el valor 34.000. Esto sucede porque el valor no está activo, bien porque no se ha acometido una transacción de memoria con el nuevo valor, o porque no se ha almacenado en la base de datos, y por tanto, muestra el valor previo por ser el que se encuentra en memoria.

Ejecución de métodos y conceptos

Los conceptos adquieren el valor que se genera a partir de la fórmula LN4 a la que estén asociados o el valor que se le asigne como si de una propiedad se tratase.

Los métodos equivalen a las funciones y procedimientos de los lenguajes de programación. Estos ejecutan una serie de instrucciones LN4 y pueden utilizarse, entre otras cosas, para actualizar los elementos de tipo campo de las estructuras de nodo que pertenecen al mismo Meta4Object u otro diferente.

Los métodos y los conceptos pueden ejecutarse de dos formas:

- Desde la interfaz de la aplicación, mediante un botón que aparece en pantalla al acceder a la información identificativa de un nodo o una presentación.
- Desde el código LN4 de otros métodos y conceptos.

  Para llamar a otros métodos o conceptos del mismo nodo o de un nodo diferente, se utilizará la sintaxis que se indica en el apartado Llamadas a los valores de los elementos de las estructuras de nodo.

  Este tipo de operaciones se pueden realizar también entre varios Meta4Objects.

Ejecución de un grupo de conceptos

En el caso de los conceptos, es posible ejecutar varios conceptos a la vez. Para ello se utilizará la función LN4 ExecuteGroup (Nombre de grupo). Al llamar a esta función, LN4 ejecutará uno tras otro todos los conceptos o métodos que estén incluidos en un mismo grupo.

Un grupo puede incluir un número ilimitado de conceptos, pero un concepto sólo puede pertenecer a un grupo, todo ello dentro de la misma estructura de nodo.

La inclusión de un concepto en un grupo se indicará al definir los elementos de la estructura de nodo desde el Diseñador de estructuras de nodo, rellenando la propiedad **Grupo de ejecución** dentro del **Editor de Meta4Objects**.

El modelo de datos actual no incluye ninguna tabla para registrar los grupos de conceptos, es decir, los valores que se indiquen en la propiedad **Grupo de ejecución** no son clave externa de ninguna otra tabla de la base de datos.

El orden en el que se ejecutarán los conceptos lo determina el programa de forma dinámica a partir de las dependencias que existan entre los conceptos y los métodos del grupo que se quiere ejecutar.

El lenguaje LN4 no permite alterar el orden de ejecución elegido por el

Meta4Object Engine. Además, se debe evitar la existencia de ciclos en el proceso, ya que se produciría un fallo y se detendría la ejecución.

Es posible ejecutar más de una vez la función ExecuteGroup (Nombre de grupo) en un mismo método. En este caso, el programa no determinará el orden en el que se tienen que ejecutar los distintos grupos. Por lo tanto, podría darse el caso de que se ejecutase en primer lugar un grupo que incluye un concepto en cuya fórmula se está haciendo referencia a un concepto cuyo cálculo se va a procesar en un grupo posterior.

El programador debe controlar estas circunstancias y establecer un orden de ejecución de grupos correcto, para evitar así posibles inconsistencias.

Funciones LN4

Argumentos de las funciones LN4

Las funciones LN4 pueden recibir de 0 a n argumentos de entrada, dependiendo de cada caso, escritos entre paréntesis y separados por comas.

Dentro de la lista de argumentos de una función LN4 se pueden incluir:

- Constantes: números, literales y fechas.
- Variables que recojan datos de diverso tipo (fecha, número, cadena).
- Llamadas a otras funciones LN4.

Para escribir correctamente la sintaxis adecuada, es necesario tener en cuenta ciertos aspectos:

- Si se indica como argumento de una función una constante de tipo cadena de caracteres, se debe delimitar el valor del argumento con comillas dobles.

EJEMPLO

```
EMPLEADOS.StrNombreelemento = "argumento"
```

- Si se indica como argumento de una función una constante de tipo número, se debe escribir el número tal cual, sin emplear ningún signo adicional. Es posible indicar el valor de este argumento utilizando el formato científico. Para más información sobre este formato, consulte el apartado *Tipos de datos.*

EJEMPLO

```
EMPLEADOS.Salario = 3000000
```

- Si se introduce como argumento una constante de tipo fecha, debe utilizarse la siguiente sintaxis:

  ```
  {yyyy-mm-dd HH:MM:SS}
  ```

  A la hora de utilizar este tipo de dato como argumento de una función, debe tener en cuenta los siguientes aspectos:

  - La fecha debe ir delimitada por llaves "{ }".

- El año debe escribirse con cuatro dígitos; para el mes, el día, las horas, los minutos y los segundos se utilizarán dos dígitos.
- El año, el mes y el día se separan mediante el carácter "-".
- Las horas, minutos y segundos se separan mediante el signo ":".
- Entre el día y la hora debe dejarse un espacio en blanco.
- Es posible omitir las horas, minutos y segundos. En este caso, el programa empleará la hora 00, el minuto 00 y el segundo 00.

- Si se introduce como argumento de una función el valor de una variable, basta con escribir el nombre de la variable.
- También es posible indicar como argumento de una función el valor de un elemento de la misma estructura de nodos o de otra estructura, ya sea del mismo Meta4Object o de otro:
  - Si el elemento es de tipo campo, la función recibirá como argumento el valor que se haya asignado a ese elemento en la base de datos.
  - Si el elemento es de tipo método, la función recibirá como argumento el valor que se obtenga al ejecutar el método que se introduzca como argumento. En este caso hay dos posibilidades:
    - Se pasa el valor del concepto al método sin ser ejecutado, mediante *método (concepto)*.
    - Se ejecuta primero el concepto y luego se pasa su valor al método, con *método (concepto( ))*.

EJEMPLO

Vamos a diseñar una fórmula que obtenga el valor máximo de dos valores. Estos dos valores se obtendrán calculando el valor absoluto de dos elementos incluidos en otras estructuras de nodo del mismo Meta4Object.

Para ello se emplea la función Max ( ), tal y como se indica en la siguiente sentencia:

```
Max ( Abs ( TI1.elemento), Abs (TI2.elemento) )
```

Tipos de funciones

Las funciones que incorpora el lenguaje LN4 permiten llevar a cabo diversas acciones sobre los registros de un nodo, por ejemplo:

- Añadir registros en una base de datos lógica (memoria)
- Borrar registros en una base de datos lógica (memoria)
- Moverse entre registros
- Buscar los registros que cumplen unas condiciones determinadas, etc.

Cada una de ellas pertenece a uno de los siguientes grupos:

- Funciones para ejecución de código JIT
- Biblioteca básica de funciones
- Funciones para Meta4Objects
- Funciones para manejo de registros
- Funciones para la base de datos lógica
- Biblioteca básica de funciones de nivel 2
- Funciones para Meta4Objects
- Biblioteca básica de funciones de nivel 2
- Funciones de log
- Funciones herramienta
- Funciones para transacciones de memoria
- Funciones de nivel 1 para metadatos
- Funciones de nómina
- Funciones de moneda
- Funciones para estadísticas
- Macros LN4

Funciones para ejecución de código JIT

La tecnología JIT (Just in Time) permite ejecutar un código determinado durante la ejecución de otro código. De ahí que las funciones que se recogen en este apartado permitan realizar operaciones de compilación de código LN4.

A esta categoría pertenecen las siguientes funciones:

- ClcExecuteLN4JIT (CódigoLN4, Arg1, ValArg1, Arg2, ValArg2...)
- ClcPrepareJIT (CódigoLN4, &Manejador, Arg1, Arg2,...)
- ClcExecuteJIT (Manejador, ValArg1, ValArg2,...)
- ClcReleaseJIT (Manejador)

ClcExecuteLN4JIT (CódigoLN4, Arg1, ValArg1, Arg2, ValArg2...)

Esta función compila y ejecuta el código LN4 que se indica como primer parámetro y que se utiliza en los distintos argumentos Arg1, Arg2, etc y los diferentes valores correspondientes a esos argumentos ValArg1, ValArg2, etc.

ClcPrepareJIT (CódigoLN4, &Manejador, Arg1, Arg2,...)

Esta función compila el código LN4 que se indica como primer parámetro, utilizando el resto de argumentos, y devuelve como resultado una referencia a la fórmula compilada, que se podrá utilizar en una ejecución posterior con la función ClcExecuteJIT (Manejador, ValArg1, ValArg2,...))

La referencia a la fórmula compilada se devuelve en &Manejador, y el carácter & significa que esta variable indica por referencia el valor correspondiente a ese argumento.

ClcExecuteJIT (Manejador, ValArg1, ValArg2,...)

Esta función ejecuta el código LN4 al que se hace referencia mediante Manejador, que ha sido previamente compilado por la función ClcPrepareJIT (CódigoLN4, &Manejador, Arg1, Arg2,...).

La función utiliza el resto de los argumentos como argumentos del código.

ClcReleaseJIT (Manejador)

Esta función borra el código LN4 al que se referencia mediante Manejador, y que se ha compilado previamente con la función ClcPrepareJIT (CódigoLN4, &Manejador, Arg1, Arg2,...).

Biblioteca básica de funciones

MessageBox (Título, Cabecera,...)

La función *MessageBox ( )* muestra un mensaje en pantalla. *MessageBox ( )* sólo puede utilizarse en la parte cliente para generar un único tipo de ventana de diálogo. La ventana incluirá un solo botón con el literal *OK*.

La función MessageBox ( ) recibe como mínimo dos argumentos. El primero se escribirá en la barra de título de la ventana y debe ser una cadena. El resto de argumentos se escribirán en el interior de la ventana, constituyendo así el cuerpo del mensaje.

EJEMPLO DE SINTAXIS

```
MessageBox (
   VALUE VARIABLE STRING Title,
```

```
   VALUE VARIABLE STRING Heading
)
```

DialogBox (Estilo, Título, Cabecera,...)

Esta función muestra un mensaje en pantalla. Los argumentos segundo y tercero deben ser una cadena de caracteres, pero no así los siguientes.

Para ver más información sobre este punto, consulte el apartado *Macros para gestionar botones en cuadros de diálogo*. Estas macros ofrecen diversas combinaciones posibles de botones que aparecerán en pantalla, por ejemplo: *Yes, No, Cancel*.

El estilo es una constante definida previamente, que define el estilo de pantalla que se muestra al usuario. Este argumento determina la presencia del botón y devuelve el valor correspondiente al botón pulsado.

El título es un argumento de tipo cadena, que contiene el título de la pantalla que se va a visualizar; mientras que la cabecera es un argumento de tipo cadena que contiene el inicio del mensaje que se muestra en pantalla.

Existen macros que permiten definir los botones que se desea que aparezcan en pantalla, y otras macros que permiten saber qué botones se han pulsado.

En una instalación en servidor, la función *Dialogbox*, al igual que la función *MessageBox*, asume que se ha hecho clic en *OK*, ya que no es lógico escribir código LN4 utilizando cualquiera de estas dos funciones para que se ejecute en la parte servidor.

En ambos casos, se muestra un aviso o error donde se indica que se asume que el usuario ha hecho clic en el botón OK.

EJEMPLO DE SINTAXIS

```
DialogBox (
   VALUE NUMBER Style,
   VALUE VARIABLE STRING Title,
   VALUE VARIABLE STRING Heading
)
```

BeginVMTransaction ( )

Esta función comienza una transacción de Meta4Object Engine.

Tipos de transacciones en Meta4Object Engine

En el Meta4Object Engine existen dos tipos de transacciones distintas:

- Por un lado, están las transacciones clásicas de base de datos, para trabajar con las cuales, el usuario dispone de las funciones *BeginDBTransaction ( )* y *EndDBTransaction (Commit/Rollback/ Execute_postvalidation)*
- Por otro lado, están las transacciones propias del Meta4Object Engine, que permiten trabajar con métodos de cuyo correcto funcionamiento no se está seguro (como el caso de la función Persist).

  En el caso de que se produzca alguna anomalía en la ejecución de estos métodos, se podrá seguir ejecutando el código.

  Las funciones para trabajar con estas transacciones son *BeginVMTransaction ( )* y *EndVMTransaction (Commit / Rollback / Execute_postvalidation)*

Existe otro conjunto de funciones que son las más usadas y que inician simultáneamente una transacción de Meta4Object Engine y de base de datos. Se trata de *BeginTransaction ( )* y *EndTransaction ( )*. Estas funciones están definidas como elementos del sistema, por lo que se puede tener acceso a ellas desde cualquier estructura de nodo.

Métodos transaccionales

Los métodos transaccionales son aquellos que inician una transacción de Meta4Object Engine antes de iniciar su ejecución, de forma que si falla algo en ese método, se detiene su ejecución, pero la del método que lo llamó sigue en la instrucción siguiente a la llamada al elemento transaccional.

Trabajar con métodos transaccionales es lo más indicado a la hora de trabajar con transacciones.

En el caso de que el elemento transaccional no funcione correctamente, se devuelve como resultado, así como en las variables por referencia, si las hay, el valor *M4_ERROR*.

Un ejemplo de método transaccional es *Persist_tree ( )*.

Posibles situaciones con transacciones de Meta4Object Engine y su resolución

En este apartado, se exponen varios ejemplos que corresponden a distintas situaciones que se pueden dar en una transacción, así como su comportamiento.

EJEMPLO 1

En este ejemplo, una vez que el método no funciona correctamente, deja de ejecutarse el código, lo que supone que no se pueda ejecutar la línea correspondiente a *MessageBox()*.

Es común pensar que se va a ejecutar a partir de la función EndTransaction, pero no es así.

En efecto, una vez que el método falla, el Meta4Object Engine se fija en el método en el que se inició la transacción para seguir posteriormente ejecutando desde el elemento del que depende.

```
BeginTransaction
        Método_que_falla()
EndTransaction (commit)
MessageBox ("el método funciona correctamente")
```

EJEMPLO 2

En este segundo ejemplo no existe otro método que llame, por lo tanto se detiene la ejecución.

```
Método_transaccional_que_falla ( )
MessageBox ("El método funciona correctamente")
```

EJEMPLO 3

En este tercer ejemplo, una vez que falla el método transaccional, sigue ejecutando por el método que lo llamo.

De este modo, ejecutaría la función MessageBox ( ).

```
Método---
        BeginTransaction()
                Método_que_falla()
        EndTransaction(commit)
        MessageBox ("El método no funciona")

MessageBox ("El método funciona correctamente")
```

En este caso, al fallar el método, se recupera hasta el anterior al que inició la transacción, por lo que el segundo MessageBox sí se ejecuta, no así el primero.

EJEMPLO 4

En este caso, se utilizan las transaccciones para abortar la ejecución de un método en el caso de que se cumpla una condición.

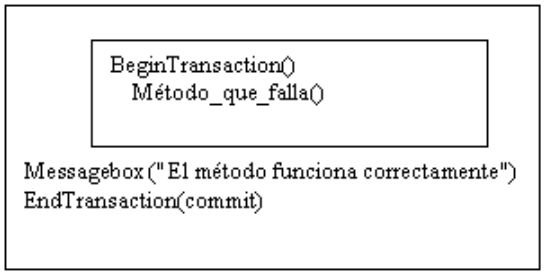
La transacción que se está cerrando en este ejemplo sin éxito, es una transacción que se abre implícitamente antes de comenzar la ejecución de ningún método, de forma que si se cierra su ejecución, se recupera el primer método antes de la llamada.

```
If count ( ) < 1 then
   SetLog (...)
   EndTransaction (Rollback)
Else
   Procesa ( )
Endif
...
```

Si este método formase parte de una cadena de llamadas y pasase por la función EndTransaction (Rollback) se comportaría como cualquier otro método que falla (por ejemplo el almacenamiento, la lectura de EOF...) y se recuperaría hasta el anterior al que inició la transacción.

EJEMPLO 5

En este caso, sí se ejecutaría el *MessageBox*, ya que al fallar, se recupera hasta el padre del elemento que empezó la transacción.



A pesar de funcionar como se espera, es poco elegante e induce a errores ya que no queda simétrica la gestión de inicio/fin de la transacción. Además, cuando se ejecuta *EndTransaction* ya no hay ninguna transacción abierta, puesto que la transacción se cerró al producirse el error, quedando como única transacción la implícita que fue abierta antes de la ejecución del primer método.

En el resto de situaciones, no se produce esta circunstancia, porque al producirse un fallo antes de ejecutar *EndTransaction*, éste no llega a ejecutarse, y el Meta4Object Engine cierra la transacción al recuperarse del error o cancela totalmente la ejecución.

EndVMTransaction (Commit/Rollback)

Finaliza una transacción de Meta4Object Engine. Recibe un único argumento, con dos posibles valores:

- Commit: Ejecuta una transacción de Meta4Object Engine.
- Rollback: Aborta una ejecución.

EJEMPLO DE SINTAXIS

```
EndVMTransaction(
        VALUE VARIABLE STRING Commit/RollBack
)
```

Null ( )

La función *Null ( )* no realiza ninguna acción.

Round (Valor, Precisión)

Esta función redondea el valor que recibe como primer argumento, utilizando la precisión que se le indica como segundo argumento.

La función se comporta de distinta forma, dependiendo del tipo de dato del valor que recibe como primer argumento:

- Si el primer argumento es un valor de tipo numérico:
  - Si la precisión es mayor que 0, se redondea la parte decimal.

EJEMPLO

La función Round (123.456, 1) devolvería el valor 123.5

La función Round (123.456, 2) devolvería el valor 123.46

La función Round (123.456, 3) devolvería el valor 123.456

  - Si la precisión es menor que 0, se redondea la parte entera.

EJEMPLO

La función Round (123.456, -1) devolvería el valor 120.

La función Round (123.456, -2) devolvería el valor 100.

La función Round (123.456, -3) devolvería el valor 0.

- Si el primer argumento es un valor de tipo cadena de caracteres:
  - Si la precisión (segundo argumento) es mayor que 0, la función devuelve los n primeros caracteres comenzando por la izquierda, siendo n el número indicado como segundo argumento en la llamada a la función.

EJEMPLO

La función Round (“Empleado”, 2) devolverá la cadena de caracteres Em.

– Si la precisión es menor que 0, la función devuelve los n primeros caracteres comenzando por la derecha, siendo «n» el número indicado como segundo argumento en la llamada a la función.

EJEMPLO

La función Round(“Empleado”, -5) devolverá la cadena de caracteres "leado".

- Si el primer argumento es un valor de tipo fecha:

  En este caso, la precisión debe estar comprendida entre 1 y 6. Si se utiliza precisión 1, la función devuelve el año; si se utiliza precisión 2, devuelve el año y el mes, y así sucesivamente.

EJEMPLO

Las siguientes funciones devuelven los siguientes valores:

Round ({ 1999-07-22 12:30:15 }, 1)	1999-01-01 00:00:00
Round ({ 1999-07-22 12:30:15 }, 2)	1999-07-01 00:00:00
Round ({ 1999-07-22 12:30:15 }, 3)	1999-07-22 00:00:00
Round ({ 1999-07-22 12:30:15 }, 4)	1999-07-22 12:00:00
Round ({ 1999-07-22 12:30:15 }, 5)	1999-07-22 12:30:00
Round ({ 1999-07-22 12:30:15 }, 6)	1999-07-22 12:30:15

Mid (Valor, Inicio, longitud)

La función *Mid ( )* extrae de la cadena de caracteres que recibe como primer argumento, los caracteres comprendidos entre las posiciones indicadas en el segundo y el tercer argumento.

La función recibe tres argumentos:

- El primero es la cadena de caracteres original.
- El segundo indica la posición a partir de la cual se comenzarán a extraer caracteres.
- El tercero indica el número de caracteres a extraer.

EJEMPLO

La función *Mid ("Cálculo de la cotización", 10, 4)* devuelve la subcadena *"e la"*, es decir, los caracteres comprendidos entre las posiciones 10 y 14, ambas incluidas.

CommonTime (InicFecha1 , Fecha de fin_1 , InicFecha$_2$, Fecha de fin_2)

Esta función devuelve el tiempo en común entre dos intervalos de tiempo. Recibe como argumentos cuatro fechas, que se corresponden con la fecha de inicio y de fin del primer y segundo intervalo de tiempo que se quiere comparar. Las fechas deben indicarse en el formato aceptado por el lenguaje LN4.

La función devuelve un valor de tipo fecha {yyyy-mm-dd hh:mm:ss}. Es importante tener en cuenta que se considera el día como unidad.

CommonTimeStamp (FechaInicio$_1$, FechaFin$_1$, FechaInicio$_2$, FechaFin$_2$)

Esta función devuelve el tiempo compartido entre dos intervalos de tiempo, usando fechas y horas. La particularidad de esta función estriba en que las diferencias de horas en el mismo día no se cuentan como un día completo. Si alguno de los argumentos es nulo esta función devuelve el valor NULL.

Format (Número, Precision)

La función *Format ( )* convierte a cadena el valor que recibe como primer argumento, poniendo como número de cifras decimales tantos caracteres como se indique en el segundo argumento.

Si la función recibe como primer argumento un número que no tiene parte decimal, se añadirán tantos 0 a la derecha del punto decimal como se hayan indicado en la precisión.

El segundo argumento que recibe la función siempre tiene que ser un número positivo. Si la precisión que se indica como segundo argumento es superior a la precisión del número que se indica como primer argumento, la función añadirá ceros a la derecha de la parte decimal.

EJEMPLO

Pongamos como ejemplo las siguientes llamadas a la función Format ( ) que devolverán los siguientes valores:

Llamadas a la función Format ( )	Valores devueltos
`Format ( 123.56, 1)`	`123.5`
`Format (1689.69, 4)`	`1689.6900`
`Format (1689.6915, 2)`	`1689.69`
`Format (8915.312, 1)`	`8915.3`

DayOfTheWeek (Fecha)

La función *DayOfTheWeek ( )* devuelve el día de la semana (lunes, martes, etc.) correspondiente a la fecha que se le indica como argumento. La función devuelve los siguientes valores:

1	Lunes
2	Martes
3	Miércoles
4	Jueves
5	Viernes
6	Sábado
7	Domingo

La fecha se puede indicar como variable, valor de un elemento, o en un formato de fecha válido para el lenguaje LN4.

EJEMPLO

Las siguientes llamadas a la función DayOfTheWeek devolverán los siguientes valores:

`DayOfTheWeek ( { 1999-07-25 } )`	`5`
`DayOfTheWeek ( { 1999-07-22 } )`	`2`
`DayOfTheWeek ( { 1999-11-23 } )`	`7`
`DayOfTheWeek ( { 1999-12-25 ) }`	`4`

Months (Fecha1, Fecha2)

La función *Months (Fecha1, Fecha2)* devuelve el número de meses transcurridos entre las dos fechas que recibe como argumentos.

El orden de los argumentos afecta al resultado de la función, es decir, si la fecha que recibe la función como primer argumento es mayor que la fecha que recibe en segundo lugar, la función devolverá valores negativos.

La función no considera el número de días naturales de los meses que se le indican como argumento. Por ejemplo, las dos llamadas a la función *Months ( )* que se indican a continuación devolverían el mismo valor a pesar de que enero y febrero no tiene el mismo número de días:

```
Months ({ 1999-01-01 },{ 1999-02-01 } )
Months ({ 1999-02-01 },{ 1999-03-01 } )
```

EJEMPLO

Las siguientes llamadas a la función Months ( ) devolverán estos valores de meses transcurridos como números enteros:

Llamadas a la función Months ( )	Valores devueltos
Months ({ 1999-01-01 },{ 1999-03-01 })	2
Months ({ 1999-05-01 },{ 1999-12-10 })	7
Months ({ 1999-03-12 },{ 1999-03-30 })	0
Months ({ 1999-03-30 },{ 1999-03-02 })	-1
Months ({ 2004-03-30 },{ 2003-03-02 })	-13

Years ($Fecha_1$, $Fecha_2$)

Esta función devuelve el número de años transcurridos entre las dos fechas que se le indican como argumentos.

El orden de los argumentos afecta al resultado de la función, es decir, si la fecha que recibe la función como primer argumento es mayor que la fecha que recibe en segundo lugar, la función devolverá valores negativos.

La función devolverá la diferencia obtenida al restar la diferencia entre el año mayor y el menor, sin considerar los meses, días, ni el resto de valores indicados en la fecha.

Por ejemplo, las dos llamadas a la función *Years* que se indican a continuación devolverán el mismo valor:

```
Years ( { 1990-01-01 },{ 1996-01-01 } )
Years ( { 1990-01-01 },{ 1996-12-01 } )
```

EJEMPLO

Las siguientes llamadas a la función *Years ( )* devolverán los valores que recogen las celdas de su derecha:

Llamadas a la función Years ( )	Valores devueltos
`Years ( { 1990-12-10 },{ 1991-01-01 })`	1
`Years ( { 1990-12-10 },{ 1996-10-10 } )`	6
`Years ( { 1996-10-09 },{ 1996-07-22 } )`	0
`Years ( { 1998-01-01 },{ 2005-02-03 } )`	7

Max ($Value_1$, $Value_2$, … , $Value_n$)

Esta función devuelve el valor más alto de todos los que recibe como argumentos.

La función puede recibir un número ilimitado de argumentos, que pueden ser de tipo numérico o fecha. Todos los valores que se le indiquen como argumentos en una llamada a la función, deben ser del mismo tipo.

La función *Max ( )* no acepta datos de tipo cadena de caracteres.

EJEMPLO

Las siguientes llamadas a la función *Max ( )* devolverán los valores que se indican en la columna de la derecha.

Llamadas a la función Max ( )	Valores devueltos
`Max ( 5, 69, 42, 35, 15 )`	`69`
`Max ( { 1999-07-22 },{ 1999-08-01 } )`	`{ 1999-08-01 }`

`A={ 1999-07-22 }`
`B={ 1999-08-01 }`
`Max (A,B)`
	`{ 1999-08-01 }`

Min ($Valor_1$, $Valor_2$, … , $Valor_n$)

Esta función devuelve el más bajo de todos los valores que recibe como argumentos.

La función puede recibir un número ilimitado de argumentos, que pueden ser de tipo numérico o fecha.

Todos los valores que se indiquen como argumentos en una misma llamada a la función deben ser del mismo tipo.

La función *Min ( )* no acepta datos de tipo cadena de caracteres.

EJEMPLO

Las siguientes llamadas a la función Min ( ) devolverán los valores que se indican en la columna de la derecha.

Llamadas a la función Min ( )	Valores devueltos
`Min ( 5, 69, 42, 35, 15 )`	`5`
`Min ( 13, -10, 22 )`	`-10`
`Min ( { 1999-07-22 },{ 1999-08-01} )`	`{ 1999-07-22 }`

`A={ 1999-07-22 }`
`B={ 1999-08-01 }`
`Max (A,B)`
	`{ 1999-07-22 }`

Abs (Fecha o Número)

La función *Abs ( )* puede recibir como argumento un valor de tipo número o fecha.

- Si recibe un valor de tipo número, *Abs ( )* devolverá el valor absoluto del número que se le asigna como argumento.
- Si recibe un valor de tipo fecha, *Abs ( )* devolverá el año, el mes y el día, e igualará a 0 las horas, los minutos y los segundos.

EJEMPLO

Las siguientes llamadas a la función *Abs ( )* devolverán los valores indicados en la columna de la derecha.

Llamadas a la función Abs ( )	Valores devueltos
`Abs (-78.15)`	`78.15`
`Abs (69 )`	`69`
`Abs ( -69.42 )`	`69.42`
`Abs ({1999-07-22 15:30:10})`	`{1999-07-22 00:00:00}`

Date30 (Fecha)

La función *Date30 ( )* devuelve la fecha que correspondería al día que se le asigne como argumento, en el caso de que todos los meses tuviesen 30 días.

La fecha que se indica como argumento debe escribirse en el formato aceptado por el lenguaje LN4 para datos de tipo fecha, variable, o elemento entre otros.

Es una función específica de nómina, que convierte una fecha a su correspondiente fecha en el calendario Juliano (es decir, número de días transcurridos desde el día 1 del año 0), y a continuación divide esta fecha entre 30 para saber cuántos meses y años habrían transcurrido si los meses tuviesen 30 días.

AddMonths (Fecha, NúmeroMeses)

La función *AddMonths ( )* suma el número de meses que se le asignan como segundo argumento a la fecha que recibe como primer argumento.

El segundo argumento puede ser un entero positivo o negativo. Si el segundo argumento es negativo, la función restará meses a la fecha que recibe como primer argumento.

EJEMPLO

Las siguientes llamadas a la función *AddMonths ( )* devolverán los valores que se indican en la columna de la derecha.

Llamadas a la función AddMonths ( )	Valores devueltos
`AddMonths ( { 1999-07-01 }, 3 )`	`{ 1999-10-01 00:00:00 }`
`AddMonths ( { 1999-01-15 22:15:00 }, 5 )`	`{ 1999-06-15 22:15:00 }`
`AddMonths ( { 1973-07-22 }, 12 )`	`{ 1974-07-22 }`
`AddMonths ( { 1998-02-02 23:59:59 }, -3 )`	`{ 1999-11-02 23:59:59 }`

AddYears (Fecha, NúmeroAños)

La función *AddYears ( )* suma a la fecha que recibe como primer argumento el número de años que se le asignan como segundo argumento.

El segundo argumento puede ser un entero positivo o negativo. Si el segundo argumento es negativo, la función restará ese número en años a la fecha que recibe como primer argumento.

EJEMPLO

Las siguientes llamadas a la función *AddYears ( )* devolverán los valores que se indican en la columna de la derecha.

Llamadas a la función AddYears ( )	Valores devueltos
`AddMonths ( { 1999-07-01 }, 3 )`	`{ 2000-07-01 }`
`AddMonths ( { 1990-01-15 22:15:00 }, 5 )`	`{ 1995-01-15 22:15:00 }`
`AddMonths ( { 1973-07-22 }, 12 )`	`{ 1985-07-22 }`
`AddMonths ( { 1998-02-02 }, -3 )`	`{ 1995-02-02 }`

Fraction (Número)

La función *Fraction ( )* devuelve la parte decimal del número que se le asigna como argumento.

EJEMPLO

Las siguientes llamadas a la función *Fraction ( )* devolverán los valores que se indican en la columna de su derecha:

Llamadas a la función Fraction ( )	Valores devueltos
`Fraction ( 12.34 )`	`0.34`
`Fraction ( -90.567 )`	`-0.567`
`Fraction ( 10.10 )`	`0.10`
`Fraction (-890.45)`	`-0.45`

Div (Número1, Número2)

Esta función permite obtener el resto de la división correspondiente a dividir el primer argumento entre el segundo. Los casos posibles son:

- Si uno de los argumentos es nulo, el resultado es nulo.
- Si existe un error, devuelve el valor M4_ERROR.
- Si el segundo argumento es 0, devuelve como resultado 0.

EJEMPLO DE SINTAXIS

```
Returns
   Div (
      VALUE NUMBER Number,
      VALUE NUMBER Number2
   )
```

Bound (Valor, limInf, limSup)

La función *Bound ( )* recibe tres argumentos, que pueden ser de tipo numérico o de tipo fecha.

El primer argumento es un valor cualquiera, y los argumentos segundo y tercero son los límites inferior y superior que delimitan un intervalo. El segundo argumento debe ser menor que el tercer argumento.

La función se comporta de la siguiente forma:

- Si el valor que se le pasa como primer argumento está comprendido en el intervalo delimitado por los argumentos que se le asignan en segundo y en tercer lugar, la función devuelve el valor del primer argumento.
- Si el valor que se le asigna como primer argumento es menor que el límite inferior del intervalo, la función devuelve el límite inferior. Es decir, el número o la fecha que ha recibido como segundo argumento.
- Si el valor que se le asigna como primer argumento es mayor que el límite superior del intervalo, la función devuelve el límite superior. Es decir, el número o la fecha que ha recibido como tercer argumento.

EJEMPLO

Las siguientes llamadas a la función Bound ( ) devolverán los valores que se indican en la columna de su derecha:

Llamadas a la función Bound ( )	Valores devueltos
`Bound ( 34, 42, 69 )`	`42`
`Bound ( 15, 42, 69 )`	`42`
`Bound ( 80, 42, 69)`	`69`
`Bound ( { 1999-07-22 }, { 1999-07-01 },{ 1999-08-01 } )`	`{ 1999-07-22 }`
`Bound ( { 1999-06-30 }, { 1999-07-01 },{ 1999-08-01 } )`	`{ 1999-07-01 }`
`Bound ( { 1999-09-10 },{ 1999-07-01 },{ 1999-08-01 } )`	`{ 1999-08-01 }`

DaysOfYear (Número)

Devuelve el número de días del año que se le indican como argumento.

Si se le asigna como argumento una constante, debe escribirse el año con cuatro cifras.

EJEMPLO

Las siguientes llamadas a la función *DaysOfYear ( )* devolverán los valores que se indican en la columna de su derecha:

Llamadas a la función DaysOfYear ( )	Valores devueltos
`DaysOfYear (1999)`	`365`
`DaysOfYear (1993)`	`365`
`DaysOfYear (1990)`	`365`
`DaysOfYear (2000)`	`366`

DaysOfMonth (Fecha)

Devuelve el número de días del mes que se le asigna como argumento.

EJEMPLO

Las siguientes llamadas a la función *DaysOfMonth ( )* devolverán los valores que se indican en la columna de la derecha:

Llamadas a la función DaysOfMonth ( )	Valores devueltos
`DaysOfMonth ( { 1999-11-01 } )`	`30`
`DaysOfMonth ( { 1999-07-15 } )`	`31`
`DaysOfMonth ( { 1999-02-20 } )`	`28`
`DaysOfMonth ( { 2002-02-01 } )`	`29`

Power (Número1, Número2)

Esta función devuelve el resultado obtenido al elevar el número que recibe como primer argumento a la potencia indicada en el número que se le asigna como segundo argumento.

EJEMPLO

Las siguientes llamadas a la función Power ( ) devolverán los valores que se indican en la columna de la derecha:

Llamadas a la función Power ( )	Valores devueltos
`Power ( 5, 2 )`	`25`
`Power ( 5, -2 )`	`1/25`
`Power ( -5, 3 )`	`-125`
`Power ( -2, -3 )`	`-1/8`

Today ( )

Devuelve la fecha del sistema. No recibe ningún argumento.

EJEMPLO

Si la fecha del sistema es:

```
22-11-1998
```

La función Today ( ) devolverá el valor:

```
{ 1998-11-22 }
```

TodayNow ( )

Esta función devuelve la fecha y hora del sistema. No recibe ningún argumento.

EJEMPLO

Si la fecha del sistema es:

```
22-11-1998 y la hora 15:45:09,
```

la función TodayNow ( ) devolverá el valor siguiente:

```
{ 1998-11-22 15:45:09 }
```

TodayGMT ( )

Devuelve la fecha del sistema en formato GMT. Puede ser de gran ayuda para la sincronización de servidores en diferentes puntos del mundo.

TodayNowGMT ( )

Devuelve la fecha y hora del sistema en formato GMT. Puede ser de gran ayuda para la sincronización de servidores en diferentes puntos del mundo.

Percentage (Porcentaje, Total)

Devuelve el valor obtenido al calcular el porcentaje que recibe como segundo argumento esta función, al número que recibe como primer argumento.

El segundo argumento puede ser cualquier número real.

EJEMPLO

Las siguientes llamadas a la función *Percentage ( )* devolverían los valores que se indican en la columna de su derecha:

Llamadas a la función Percentage ( )	Valores devueltos
`Percentage ( 100, 20 )`	`20`
`Percentage ( 1000, 30 )`	`300`
`Percentage ( 250, 20 )`	`50`
`Percentage ( -300, 5 )`	`-15`

ToInt (Valor)

Esta función devuelve la parte entera de un número con parte decimal, sin realizar ningún tipo de redondeo.

Si la función recibe como argumento un dato que no es de tipo número, por ejemplo una fecha o una cadena de caracteres, intentará convertirla a un número entero. Si no consigue convertir el valor, devolverá el valor 0.0.

EJEMPLO

Las siguientes llamadas a la función ToInt ( ) devuelven el valor que se indica en la columna de su derecha.

Llamadas a la función ToInt ( )	Valores devueltos
`ToInt ( 23.56 )`	`23.00`
`ToInt ( 32.99 )`	`32.00`
`ToInt ( -15.24 )`	`-15.00`
`ToInt ( 27.986 )`	`27.00`

ToDouble (Valor)

Esta función convierte el valor que recibe como argumento a un número en doble precisión, con parte entera y decimal.

Por defecto, el programa utiliza seis cifras en la parte decimal a la hora de la impresión o visualización en pantalla.

EJEMPLO

Las siguientes llamadas a la función ToDouble ( ) devolverían los valores que se indican en la columna de su derecha:

Llamadas a la función ToDouble ( )	Valores devueltos
`ToDouble ( 23 )`	`23.000000`
`ToDouble ( 89.6 )`	`89.600000`
`ToDouble ( -21 )`	`-21.000000`
`ToDouble ( 90 )`	`90.000000`

ToString (Valor, [Precisión])

Esta función convierte a cadena de caracteres el número o fecha que recibe como primer argumento.

El segundo argumento es opcional, y dependiendo del tipo del primer argumento contendrá datos distintos:

- Si el primer argumento es un número, contiene el número de decimales del número.
- Si el primer argumento es una fecha, indica si es formato corto o largo.

Aunque LN4 gestiona los tipos de datos de forma dinámica, en ocasiones puede ser conveniente realizar conversiones entre tipos.

Por ejemplo, si se quieren concatenar los valores de dos variables que recogen datos de tipo numérico para formar una única cadena de caracteres, será necesario convertir antes las variables al tipo de dato Cadena, ya que LN4 utiliza el mismo operador para concatenar cadenas y para sumar números.

Si la conversión no se realiza con éxito, la función devolverá el valor 0.0.

EJEMPLO

Las siguientes llamadas a la función ToString ( ) devolverán los valores que se indican en la columna de su derecha:

Llamadas a la función ToString ( )	Valores devueltos
`ToString ( 34.3 )`	`"34.3"`
`ToString ( -13.5 )`	`"-13.5"`
`ToString ( { 1999-07-22 11:30:00 } )`	`"1999-07-22 11:30:00"`
`ToString ( 45 )`	`"45"`

ToDate (Valor)

Convierte la cadena de caracteres o el número que recibe como argumento a un dato de tipo fecha.

EJEMPLO

Las siguientes llamadas a la función ToDate ( ) devolverán los valores que se indican en la columna de la derecha.

Llamadas a la función ToDate ( )	Valores devueltos
`ToDate ( "1999-07-22" )`	`{ 1999-07-22 }`
`ToDate ( "1990-12-01 15:00:00" )`	`{ 1990-12-01 15:00:00 }`
`ToDate ( "1992-10-01 )`	`{ 1992-10-01 }`
`ToDate ( "1990-08-12 10:00:00" )`	`{ 1990-08-12 10:00:00 }`

Truncate (Valor, Precisión)

Trunca la parte decimal del número que recibe como primer argumento, tantas posiciones como se indiquen en el segundo argumento, que debe ser un número entero. Se comienza a truncar por la primera cifra decimal del número.

EJEMPLO

Las siguientes llamadas a la función Truncate ( ) devolverán los valores que se indican en la columna de la derecha.

Llamadas a la función Truncate ( )	Valores devueltos
`Truncate ( 123.456 , 2 )`	`123.45`
`Truncate ( 12.67 , 1 )`	`12.6`
`Truncate ( 89, 2 )`	`89.00`
`Truncate ( 90.34 , 4 )`	`90.3400`

DateToNumbers (Fecha, var_1, var_2, var_3)

Esta función recibe cuatro argumentos: una fecha y tres variables. Como resultado, divide la fecha que recibe como primer argumento en años, meses y días, y asigna estos valores a las variables que se han indicado en segundo, tercer y cuarto lugar respectivamente. Estas variables recogerán datos de tipo entero.

Es necesario haber declarado previamente las variables que se indican como argumentos, ya que son variables por referencia. Además de esta asignación, *DateToNumbers ( )* devolverá el valor M4_SUCCESS si se ejecutó correctamente o en caso contrario M4_ERROR.

EJEMPLO

Las siguientes llamadas a la función DateToNumbers ( ) realizarían las siguientes asignaciones:

Fecha - Argumento 1	Argumento 2	Argumento 3	Argumento 4
{ 1999-12-25 }	1999	12	25
{ 1990-05-03 }	1990	5	3
{ 1992-11-11 00:12:12 }	1992	11	11
{ 1982-06-21 10:56:13 }	1982	6	21

DateToAllNumbers (Fecha, var_1, var_2, var_3, var_4, var_5, var_6)

Esta función recibe como argumento una fecha y seis variables por referencia, que obligatoriamente se deben haber declarado antes de llamar a la función.

Como resultado, se divide la fecha que recibe como primer argumento en años, meses, días, horas, minutos y segundos, y se asignan estos valores a las variables que se han indicado en las posiciones segunda a séptima, respectivamente.

EJEMPLO

Las siguientes llamadas a la variable DateToAllNumbers ( ) realizarán las siguientes asignaciones:

Fecha	Arg 1	Arg 2	Arg 3	Arg 4	Arg 5	Arg 6
{ 1999-07-22 03:50:20 }	1999	7	22	3	50	20
{ 1990-05-03 00:00:00 }	1990	5	3	0	0	0
{ 1992-11-11 00:12:12 }	1992	11	11	0	12	12
{ 1982-06-21 10:56:13 }	1982	6	21	10	56	13

NumbersToDate ($Número_1$, $Número_2$, …, $Número_6$)

Convierte a formato fecha los números que recibe como argumento. La función puede recibir un máximo de seis argumentos, cuyo orden se corresponde con el año, mes, día, hora, minutos y segundos de la fecha que se quiere crear.

No es necesario indicar a la función los seis argumentos. Basta con introducir el primer argumento, es decir, el correspondiente al año.

De todas formas, cuando se llame a la función y no se le asignen todos los argumentos, será necesario introducir aquéllos que preceden al argumento que se omite (no es posible introducir un argumento que ocupe un lugar posterior). Es decir, es posible llamar a la función *NumbersToDate ( )* introduciendo el año y el mes, pero no será posible llamar a la función si se introduce el año y día, sin indicar el mes que se trata.

En el caso de que no se asigne a la función todos los argumentos, la función realizará las siguientes conversiones:

Año - Argumento 1	Obligatorio
Mes - Argumento 2	01 - Enero
Día - Argumento 3	01 - Primer día del mes
Hora - Argumento 4	00
Minuto - Argumento 5	00
Segundo - Argumento 6	00

EJEMPLO

Las siguientes llamadas a la función NumbersToDate ( ) devolverían estas fechas:

Llamadas a la función NumbersToDate ( )	**Valores devueltos**
`NumbersToDate ( 1999, 7, 22, 12, 30, 5 )`	`{ 1999-07-22 12:30:05 }`
`NumbersToDate ( 1996, 12, 5, 0, 0, 0 )`	`{ 1996-12-05 00:00:00 }`
`NumbersToDate ( 1996, 12, 5 )`	`{ 1996-12-05 00:00:00 }`
`NumbersToDate ( 1996, 3 )`	`{ 1996-03-01 00:00:00 }`
`NumbersToDate ( 1998 )`	`{ 1998-01-01 00:00:00 }`
`NumbersToDate ( 1998, 12, 5, 6 )`	`{ 1998-12-05 06:00:00 }`

StrIn ($Cadena_1$, $Cadena_2$)

Esta función busca la cadena de caracteres que se le asigna como segundo argumento dentro de la cadena de caracteres que recibe como primer argumento. Si la cadena que recibe como segundo argumento está incluida en la que recibe como primer argumento, la función devolverá la posición de la cadena. Si no está incluida, devolverá M4_ERROR.

Si se llama a la función asignándole como argumentos datos de tipo número o fecha, *StrIn* convertirá estos argumentos en una cadena de caracteres, sin que se produzca ningún error.

EJEMPLO

Las siguientes llamadas a la función StrIn ( ) devolverán los valores que se indican en la columna de la derecha:

Llamadas a la función StrIn ( )	Valor devuelto
`StrIn ( "nómina", "casa" )`	M4_ERROR
`StrIn ( "nómina", "mina" )`	Devuelve la posición de la cadena
`StrIn ( "salario base", "salario" )`	Devuelve la posición de la cadena
`StrIn ( "redondeo a 30 días", 30 )`	Devuelve la posición de la cadena
`StrIn ("1999-03-01 es la fecha", {1999-03-01} )`	Devuelve la posición de la cadena
`StrIn ( "cotización", "salario" )`	M4_ERROR
`StrIn ( "cotización", 30 )`	M4_ERROR

NullValue ( )

Asigna el valor NULO a una variable, de modo que todas las operaciones que se hagan posteriormente con ésta devolverán el valor NULO, independientemente del tipo de dato que recogiese la variable. La función no recibe argumentos, y su sintaxis es:

```
Variable = NullValue ( )
```

EJEMPLO

```
Variable = NullValue ( )  'Se asigna a Variable el valor NULO.
Salario= 200000           'Se asigna a Salario el valor 200.000
Total = Salario + Varialbe    'Se suma el valor de Salario e Variable
' como Variable recoge el valor NULO, la suma ' ' devolverá NULO
MsgBox ("Meta4", "El total es: " + Str ( Total ) ) ' La ventana de diálogo muestra el mensaje "El total es"
```

InitFile (Fichero, Valor, [AddEOL])

Crea un archivo de texto plano, y le añade los caracteres que se le pasan a la función como segundo argumento. Como primer argumento debe indicarse el nombre del archivo y el directorio donde se encuentra en el que cual se desean grabar los datos.

El segundo argumento puede ser una constante, una variable o un dato de tipo número o fecha. La función convertirá este valor en una cadena de caracteres antes de escribir cualquier valor en el archivo.

La función siempre debe recibir los dos argumentos, y no es posible crear un archivo sin escribir ningún valor en él.

EJEMPLO

En este ejemplo la llamada a la función *InitFile ( )* creará un nuevo archivo en el subdirectorio \Meta4 de la unidad de disco C, con el nombre LOG.TXT, y escribirá en él el valor que se ha asignado a la variable *StrArchivo*.

```
StrArchivo = "Archivo de Log. Contiene cambios"
InitFile ( "C:\META4\LOG.TXT", StrArchivo )
```

ConcatFile (Fichero, Valor, [AddEOL])

Esta función abre el archivo de texto plano cuyo nombre se indica en el primer argumento y le añade los caracteres que se indican como segundo argumento.

El primer argumento debe recoger la ruta completa o relativa de un archivo existente: unidad de disco, lista de directorios, nombre y extensión. Toda esta información se pasará a la función como una cadena de caracteres.

El segundo argumento puede recoger una constante o una variable. Si se introduce un valor de tipo fecha o de tipo número, la función convertirá estos argumentos a una cadena de caracteres antes de modificar el archivo.

El tercer argumento es opcional, y permite añadir una línea después de escribir en el archivo. Esta variable puede tomar los valores *True* o *False* (por defecto es *True*).

EJEMPLO

La siguiente llamada a la función ConcatFile ( ) abre el archivo C:\Meta4\log.txt, y concatena la frase “El salario base asciende a 98.000”.

```
InitFile ( “c:\meta4\log.txt”, “Resumen salarial”)
StrInt = “El salario base asciende a 98.000”
ConcatFile ( “c:\meta4\log.txt”, StrInt )
```

OpenFile (NombreFichero, &Manejador)

Esta función abre el archivo indicado como primer argumento, pasándole el nombre de una variable por referencia, donde devolverá el identificador que se tendrá que usar posteriormente para leer del archivo y cerrarle.

El tipo puede ser:

– Número: indica el número de dígitos que tiene que leer.

– Carácter: indica que leerá los caracteres hasta encontrar el indicado, es decir, actúa de limitador.

  Por ejemplo, puede ser de interés buscar el caracter Newline para detectar los finales de párrafos.

EJEMPLO DE SINTAXIS

```
OpenFile (
   VALUE VARIABLE STRING FileName,
   REFERENCE NUMBER Handle
)
```

CloseFile (Manejador)

Cierra el archivo al que hace referencia un identificador (Manejador).

EJEMPLO DE SINTAXIS

```
CloseFile (
   VALUE NUMBER Handle
)
```

ReadFile (Manejador, Tipo, &CadenaDevuelta)

Lee del archivo indicado por el primer argumento, asignando una cadena de caracteres al tercer argumento, en función del tipo indicado por el segundo:

- Si es de tipo número, indica el número de caracteres que se van a leer del archivo.
- Si es de tipo carácter, lee hasta el carácter buscado.

EJEMPLO DE SINTAXIS

```
ReadFile (
  VALUE NUMBER Handle,
  VALUE VARIANT Type,
  [REFERENCE VARIABLE STRING StringReturn]
  )
```

ClearFiles ( )

Esta función elimina todos los archivos temporales que aparezcan como abiertos en el sistema del gestor de archivos.

DestroyFile (NombreFichero)

Borra físicamente un archivo del disco, devolviendo un código de error en caso de que no pueda llevar a cabo la operación.

GetRunningStartDate ( )

Devuelve la fecha de inicio de ejecución del tramo activo que se está ejecutando en ese momento. La fecha no se puede cambiar, ya que viene definida por el propio proceso de ejecución de tramos.

Esta función es muy útil para fines de depuración, ya que devuelve la regla concreta que se está ejecutando en ese momento. Esto puede ser debido a una de las siguientes causas:

- Se hace referencia a un elemento que tiene tramos.
- Se está trabajando con tramos del sistema de asignación de valores.
- Se tiene más de una norma para la misma regla, etc.

Por ejemplo, en caso de que en la ejecución en tramos sea necesario que el elemento se ejecute 3 veces; si en el código del elemento está incluida la función GetRunningStartDate, se mostrará la fecha de inicio de cada tramo que se ejecute.

Esto puede ser interesante para conocer la siguiente información:

- Para conocer por qué se está ejecutando alguna función más de lo esperado.
- El periodo en el que se está ejecutando.
- El origen de alguna dependencia, etc.

GetRunningEndDate ( )

Similar a la función *GetRunningStartDate*(), esta función devuelve la fecha de fin de ejecución del tramo activo que se está ejecutando en ese momento. La fecha no se puede cambiar, ya que viene definida por el propio proceso de ejecución de tramos.

Esta función es muy útil para la depuración, ya que devuelve la regla concreta que se está ejecutando en ese momento. Por ejemplo, en caso de que en la ejecución en tramos sea necesario que el elemento se ejecute 3 veces; si en el código del elemento está incluida la función GetRunningEndDate, se mostrará la fecha de fin de cada tramo que se ejecute.

GetRunningRunDate ( )

La función GetRunningRunDate devuelve la fecha en la que se llevó a cabo la ejecución del tramo activo o la fecha de corte del tramo que se está ejecutando en ese momento, ya que generalmente se trabaja con dos tramos. Estas fechas no se pueden cambiar, ya que vienen definidas por el propio proceso de ejecución de tramos.

Otra utilidad de esta función consiste en conocer cuál de los dos tramos es el activo en un momento determinado.

Esta función es muy útil para la depuración, ya que devuelve la regla concreta que se está ejecutando en ese momento. Por ejemplo, en caso de que en la ejecución en tramos sea necesario que el elemento se ejecute 3 veces; si en el código del elemento está incluida la función GetRunningRunDate, se mostrará la fecha de ejecución o fecha de corte de cada tramo que se ejecute.

La fecha de corte se utiliza cuando se trabaja con métodos, no así las dos

anteriores (fecha de inicio y fin) que se utilizan normalmente para conceptos, ya que se ejecutan desde distintos periodos.

GetStartDate ( )

Esta función devuelve la fecha de inicio del periodo que se está ejecutando. La función *GetAppStartDate* realiza las mismas funciones.

GetEndDate ( )

Esta función devuelve la fecha de fin del periodo que se está ejecutando. La función *GetAppEndDate* realiza las mismas funciones.

GetRunDate ( )

Esta función devuelve la fecha de ejecución del periodo. La función *GetAppRunDate* realiza las mismas funciones.

SetStartDate (Fecha)

Esta función asigna la fecha de inicio de un periodo, mediante el argumento que recibe. La función *SetAppStartDate* realiza las mismas funciones. Si la fecha es nula, devuelve como resultado el valor nulo.

EJEMPLO DE SINTAXIS

```
SetStartDate (
        VALUE DATE AND TIME Date
)
```

SetEndDate (Fecha)

Esta función asigna la fecha de fin de un periodo, mediante el argumento que recibe. La función *SetAppEndDate* realiza las mismas funciones. Si la fecha es nula, devuelve como resultado el valor nulo.

EJEMPLO DE SINTAXIS

```
SetEndDate (
   VALUE DATE AND TIME Date
)
```

SetRunDate (Fecha)

Esta función asigna la fecha de ejecución o corte, mediante el argumento que recibe. Si la fecha es nula, esta función asigna el valor nulo.

EJEMPLO DE SINTAXIS

```
SetRunDate(
   VALUE DATE AND TIME Date
)
```

IsNull (var)

Esta función recibe como argumento una variable y su ejecución puede devolver dos posibles resultados:

- Si el valor que recibe como argumento contiene un valor nulo, devuelve el valor M4_TRUE.
- Si el valor que contiene la variable o el campo no es nulo, devuelve el valor M4_FALSE.

EJEMPLO

```
/* Este ejemplo muestra un mensaje en pantalla */
/* dependiendo de si la variable recoge un valor nulo o no*/
If IsNull ( Nombre)=M4_TRUE Then
     MessageBox ("Titulo", Nombre + " contiene un valor
     nulo")
Else
     MessageBox("Título", "El valor de Nombre es " + Nombre)
EndIf
```

Length (var)

Esta función devuelve como resultado la longitud de una cadena, que puede ser de tipo texto, fecha o número.

Esta función puede devolver diferentes resultados:

- Si el valor es NULO, devuelve NULL.
- Si el valor es un número, se convierte a una cadena con 8 decimales.

EJEMPLO DE SINTAXIS

```
Length (
   VALUE VARIANT Value
)
```

Trim (Cadena, Dirección)

Esta función elimina los espacios en blanco situados antes o después de una cadena. Sus argumentos son los siguientes:

- Cadena

  Contiene la cadena origen de la cual se tienen que eliminar los espacios en blanco existentes antes o después, en función del argumento *Dirección*.

  Tenga en cuenta que si la cadena es nula, la función devuelve el valor nulo.

- Dirección

  Se indica el lado de la cadena donde se deben eliminar los espacios en blanco.

  Se deben tener en cuenta los siguientes aspectos:

  - Si la dirección es nula, devuelve la cadena.
  - Si la dirección no es válida, devuelve la misma cadena.

  Este argumento puede tomar los siguientes valores:

  - LEFT: Se eliminan los espacios que existan en el lado izquierdo de la cadena.
  - RIGHT: Se eliminan los espacios que existan en el lado derecho de la cadena.
  - ALL: Se eliminan los espacios que existan a ambos lados de la cadena.

EJEMPLO DE SINTAXIS

```
Trim (
   VALUE VARIABLE STRING String,
   VALUE NUMBER Direction
)
```

ConvertCase (Cadena, Low_up)

Esta función convierte una cadena cualquiera a mayúsculas o minúsculas. Esto se debe a una costumbre muy extendida de convertir cualquier información que se incluye en la base de datos a mayúsculas, ya que las comparaciones son más sencillas y no hay que utilizar ninguna de las constantes establecidas del tipo *M4_Case*.

Los argumentos correspondientes a la función son:

- Cadena

  Contiene la cadena que se desea convertir a mayúsculas o minúsculas.

  Si la cadena indicada es nula, esta función devuelve como resultado un valor nulo.

- Low_up

  Indica el tipo de conversión que se desea realizar sobre la cadena, y puede tomar los siguientes valores:

  – LowerCase

  Convierte la cadena a minúsculas.

  – UnChanged

  La cadena no tiene ningún cambio.

  – UpperCase

  Convierte la cadena a mayúsculas.

  Se deben tener en cuenta además los siguientes aspectos:

  – Si el contenido de este argumento es nulo, la función devuelve como resultado la cadena.
  – Si el contenido de este argumento no es válido, devuelve la cadena sin realizar ningún cambio.

Para más información sobre este tipo de constantes, consulte el apartado *Macros LN4*.

EJEMPLO DE SINTAXIS

```
ConvertCase (
   VALUE VARIABLE STRING String,
   VALUE NUMBER Low_up
)
```

Ascii (Cadena)

Esta función devuelve el código ASCII del carácter que aparece en primer lugar del alrgumento.

Su funcionamiento es opuesto a la función *Chr (Código)*.

EJEMPLO DE SINTAXIS

```
Ascii (
   VALUE VARIABLE STRING String
)
```

Chr (Código)

Esta función devuelve un valor de tipo cadena que corresponde al carácter cuyo código ASCII es el argumento especificado. Si este código es nulo, la función devuelve como resultado un valor nulo.

El funcionamiento de *Chr ( )* es opuesto a la función *Ascii (Cadena)*.

Existen una serie de constantes definidas para poder utilizarlas con la función Chr ( ) y que se enumeran a continuación:

- M4_New_Line
- M4_Tab
- M4_Double_Quote
- M4_Cr

Para obtener más información sobre este tipo de constantes, consulte el apartado *Macros de constantes CHR*.

EJEMPLO DE SINTAXIS

```
Chr (
   VALUE NUMBER Code
)
```

Replace (Cadena, SubcadenaAntigüa, SubcadenaNueva)

Esta función reemplaza una subcadena perteneciente a una cadena concreta por otra subcadena nueva que ha sido indicada como argumento.

La función devuelve la cadena obtenida como resultado de las operaciones o, en caso de producirse algún error, devolverá como resultado el valor *M4_ERROR*.

La función recibe tres argumentos, que corresponden respectivamente a la cadena original, la subcadena que se va a sustituir, y la subcadena que se va a insertar en lugar de la anterior.

Se deben tener en cuenta además los siguientes aspectos:

- Si la cadena que se va a modificar es NULL, devuelve NULL.
- Si la subcadena que se va a eliminar es NULL, devuelve la cadena.
- Si la subcadena que se va a añadir es NULL, devuelve la cadena.

EJEMPLO DE SINTAXIS

```
Replace (
   VALUE VARIABLE STRING String,
   VALUE VARIABLE STRING OldSub,
   VALUE VARIABLE STRING NewSub
)
```

IndexOf (Cadena, Subcadena, ÍndiceInicial)

Esta función busca la subcadena que se indica como segundo argumento, dentro de la cadena que se indica como primer argumento, utilizando un índice definido que se pasa como tercer argumento.

La búsqueda empieza desde el principio de la cadena, en la posición indicada por el índice, y se dirige hasta el final. Si este tercer argumento es nulo, tomará como valor el 0.

La función devuelve un resultado distinto dependiendo de los valores de los argumentos:

- Si existe algún error en su ejecución, devuelve el valor *M4_ERROR*.
- Si la cadena es nula, devuelve como resultado un valor nulo.
- Si la subcadena es nula, devuelve como resultado *M4_ERROR*.

Por ejemplo, para buscar la cadena "p" dentro de una cadena especificada a partir de la posición 5, la función devuelve el número donde se encuentra la subcadena dentro del índice.

EJEMPLO DE SINTAXIS

```
IndexOf (
   VALUE VARIABLE STRING String,
   VALUE VARIABLE STRING SubString,
   VALUE NUMBER InitialIndex
)
```

LastIndexOf (Cadena, SubCadena, [ÍndiceInicial])

Esta función, al igual que *IndexOf (Cadena, Subcadena, ÍndiceInicial)*, busca una subcadena dentro de una cadena utilizando un índice definido. La diferencia entre estas dos funciones estriba en que *LasIndexOf* empieza la búsqueda por el final de la cadena.

Los argumentos tienen idéntico significado a los de IndexOf(), y al igual que en ésta, la función devuelve un resultado distinto dependiendo de los valores de los argumentos:

- Si existe algún error en su ejecución, devuelve el valor *M4_ERROR*.
- Si la cadena es nula, devuelve como resultado un valor nulo.
- Si la subcadena es nula, devuelve como resultado *M4_ERROR*.

Por ejemplo, si se busca la cadena "p" dentro de una cadena especificada a partir de la posición 5, la función devuelve el número donde se encuentra la subcadena dentro del índice.

EJEMPLO DE SINTAXIS

```
LastIndexOf (
   VALUE VARIABLE STRING String,
   VALUE VARIABLE STRING SubString,
   [VALUE NUMBER InitialIndex]
)
```

GetCsTimeOut (Tiempo, Tipo)

Esta función indica el tiempo límite que tiene una transacción cliente-servidor, de tal forma que si se supera ese tiempo, se indica al cliente que se ha superado y se le pregunta si desea abortar la transacción.

El primer argumento indica en segundos el tiempo límite que se ha fijado para realizar una transacción completa.

El segundo indica el tipo de tiempo límite, y puede tomar los siguientes valores:

- M4_DEFAULT_CS_TIMEOUT (0)

  Ajusta el valor para los tiempos límite de ejecución OLTP/PROXY. Éste es el valor por defecto.
- M4_NEXT_CS_TIMEOUT (1)

  Ajusta el valor del tiempo límite para la siguiente transacción. Una vez ejecutada, vuelve al valor por defecto.
- M4_GENERIC_CS_TIMEOUT (2)

  Ajusta el valor para las transacciones que no son ni PROXY ni OLTP. Ejemplo: Logon, metadatos, espacios proxy, etc.

EJEMPLO DE SINTAXIS

```
GetCSTimeOut (
   VALUE NUMBER, Type
)
```

SetCsTimeOut (Tiempo, Tipo)

Esta función modifica el tiempo límite que tiene una transacción cliente/servidor para ejecutarse, de tal forma que si se supera ese tiempo, se indica al cliente que se ha sobrepasado y se le pregunta si desea abortar la transacción.

El primer argumento indica el tiempo límite que se ha fijado para realizar una transacción completa, facilitado en segundos.

El segundo argumento es opcional y puede tomar los siguientes valores:

- M4_DEFAULT_CS_TIMEOUT (0)

  Ajusta el valor para los tiempos de espera de ejecución OLTP/PROXY. Éste es el valor por defecto.
- M4_NEXT_CS_TIMEOUT (1)

Ajusta el valor para la siguiente transacción. Una vez ejecutada, vuelve al valor por defecto.

– M4_GENERIC_CS_TIMEOUT (2)

  Ajusta el valor para las demás transacciones que no son ni PROXY ni OLTP. Ejemplo: Logon, metadatos, espacios proxy, etc.

EJEMPLO DE SINTAXIS

```
SetCSTimeOut (
   VALUE NUMBER Time,
   VALUE NUMBER Type
)
```

ConfigCSTimeOut (Tipo, Marca)

Esta función configura internamente el tiempo límite de transacción. Los argumentos que recibe son los siguientes:

- Tipo

  El tipo por defecto es M4_RESET_NEXT_TIMEOUT. Este argumento puede tomar dos valores más:

  - 0: Reinicia el siguiente tiempo límite por defecto a Tipo=1.
  - 1:Habilita/Deshabilita la ventana que indica el tiempo límite.

- Marca

  Este argumento puede tomar dos valores:

  - 0: Deshabilita la ventana que indica el tiempo límite.
  - 1: Habilita la ventana que indica el tiempo límite.

A continuación se explica la posible combinación de sus argumentos:

Type = M4_RESET_NEXT_TIMEOUT **Flag = M4_TRUE**
Es equivalente a un SetCsTimeout con el valor por defecto. Se incluye por comodidad.

Type = M4_RESET_NEXT_TIMEOUT **Flag = M4_FALSE**
No realiza nada, se incluye por completitud.

Type = M4_ENABLE_TIMEOUT_WINDOW **Flag = M4_TRUE**
Permite que aparezca la ventana que indica el tiempo límite.

Type= M4_ENABLE_TIMEOUT_WINDOW **Flag = M4_FALSE**
Elimina la ventana de tiempo de espera.

EJEMPLO DE SINTAXIS

```
ConfigCSTimeOut (
   VALUE NUMBER Type,
   [VALUE NUMBER Flag]
)
```

SetCSTimeoutFunction (NombreDLL, NombreFunción)

Esta función se utiliza para indicar qué función de una determinada DLL se quiere ejecutar en el caso de que se supere el tiempo definido como límite. El primer argumento indica la DLL a la que pertenece la función que se quiere ejecutar, mientras que el segundo contiene la función propiamente dicha.

La función exportada por la DLL tiene que tener el siguiente prototipo:

```
typedef m4int32_t(*ClTimeoutFunction)(m4int32_t);
```

La variable ***m4int32_t*** es un entero con signo de 4 bytes, cuyo argumento es el valor del *timeout* actual en milésimas de segundo.

El valor de retorno puede ser:

- Menor que 0 : Indica que la ejecución se aborta.
- Igual a 0 : Indica que el tiempo de espera es infinito o que no existe un tiempo máximo.
- Mayor que 0 : Indica el tiempo para el siguiente tiempo de espera.

EJEMPLO DE SINTAXIS

```
SetCSTimeoutFunction (
   VALUE VARIABLE STRING Function,
   VALUE VARIABLE STRING DllName
)
```

GetServerValue (DirectorioOBL, VarNombre)

Esta función lee una variable de configuración correspondiente al servidor.

Recibe como argumentos la ubicación dentro del servidor de la variable de configuración que se va a leer, y el nombre de dicha variable de configuración.

EJEMPLO DE SINTAXIS

```
GetServerValue (
   VALUE FIXED STRING Path,
   VALUE FIXED STRING Variable
)
```

GetRegistryValue (DirectorioRegistro, VarNombre)

Esta función lee una variable de configuración del registro.

Recibe como argumentos la ubicación dentro del registro de la variable de configuración que se va a leer, y el nombre de dicha variable de configuración.

EJEMPLO DE SINTAXIS

```
GetRegistryValue (
   VALUE FIXED STRING Path,
   VALUE FIXED STRING Variable
)
```

GetVariantType (var)

Esta función dice si un valor que se ha indicado como argumento es de tipo cadena, fecha, número o nulo. Es útil en aquellas ocasiones en las que se indique como argumento un valor cuyo tipo se desconozca, para, en función de ello, actuar de un modo u otro.

EJEMPLO DE SINTAXIS

```
GetVariantType (
   VALUE VARIANT Value
)
```

BitwiseOr (Var1, Var2)

Esta función utiliza la operación lógica OR para realizar operaciones entre dos números que previamente se han convertido a bits.

BitwiseXor (Var1, Var2)

Esta función utiliza la operación lógica XOR para realizar operaciones entre dos números que previamente se han convertido a bits.

BitwiseAnd (Var1, Var2)

Esta función utiliza la operación lógica AND para realizar operaciones entre dos números que previamente se han convertido a bits.

BitwiseNot (Var1)

Esta función realiza la operación lógica NOT (es decir, la inversión de valores) sobre los bits de un número.

GetUniqueID ( )

Esta función genera un identificador único de 20 caracteres.

Rnd (Max)

Esta función genera un valor aleatorio entre cero y el valor del argumento que se especifique.

MatchRegExp (Cadena, ExpReg, [SensibleMayúsculasMinúsculas])

Esta función indica si una cadena establecida como primer argumento coincide con una expresión regular que se le indique como segundo argumento.

El tercer argumento es opcional, e indica con *True* o *False* si se quiere que esta función sea sensible o no a mayúsculas y minúsculas. El comportamiento por defecto es que sí lo es.

La función devuelve el índice del primer carácter que cumpla con la expresión regular. Si no lo cumple, devolverá *M4_ERROR*.

DateAdd (Fecha1, Unidades, TipoUnidad)

Esta función muestra el resultado final de sumar un cierto valor a una fecha indicada como primer argumento. El valor de incremento se indica en el segundo argumento, mientras que el tercero indica el tipo de unidad en el que está expresado dicho valor. Como unidad de incremento se pueden utilizar segundos, minutos, horas, días, semanas, meses o años.

Los comportamientos especiales de esta función son:

- Un argumento nulo devuelve resultado nulo.
- Un argumento incorrecto se intenta convertir a correcto y proseguir. Si falla la conversión, se produce un error y se detiene la ejecución, al igual que el resto de funciones.

EJEMPLO

Fecha 1	Unidad	Tipo de unidad	Resultado
1998-01-15	1	Año	1999-01-15
1996-02-29	1	Año	1997-02-28
1997-01-15	1	Mes	1997-02-15
1996-02-29	1	Mes	1996-02-29
1996-01-30	1	Mes	1996-02-28
1996-01-01	20	Día	1996-01-21
1999-01-01 23:15:00	2	Hora	1999-01-02 01:15:00
1999-01-01 23:15:00	80	Minuto	1999-01-02 00:35:00
1999-09-22	1	Semana	1999-01-29

DateDiff (Fecha1, Fecha2, TipoUnidad, [InicioSemana])

Calcula la diferencia en unidades (por ejemplo, segundos) entre dos fechas indicadas como argumentos, restándose la segunda de la primera.

El tercer argumento indica el tipo de unidad que se va a considerar, mientras que el último es un argumento opcional que indica el día que comienza la semana. Este argumento hace que la función reste semanas completas, y puede contener cualquier valor comprendido entre 1 y 7, considerándose el 1, lunes, como valor por defecto.

Los comportamientos especiales de la función son los siguientes:

- Si existe algún argumento nulo, la función devuelve como resultado un valor nulo.
- Si el tipo de argumento indicado es incorrecto, la función intenta convertirlo a correcto y seguir. Si falla en la conversión, se produce un error y se detiene la ejecución.
- Si la segunda fecha es anterior a la primera, la función devuelve un número de unidades negativo.

El tipo de unidad *M4_Complete_Week* devuelve el número de semanas completas en el que está el periodo de tiempo pasado como argumento.

Las unidades que se pueden utilizar con esta función se muestran a continuación. Para obtener más información sobre ellas, consulte el apartado *Macros LN4*:

Unidades
M4_YEAR
M4_MONTH
M4_DAY
M4_HOUR
M4_MINUTE
M4_SECOND
M4_WEEK
M4_COMPLETE_WEEK

EJEMPLO

Fecha 1	Fecha 2	TipoUnidad	Resultado
1998-01-01	1997-01-15	M4_YEAR	1
1996-02-29	1995-02-28	M4_YEAR	1
1999-03-30	1999-02-28	M4_MONTH	1
1999-04-30	1999-03-30	M4_MONTH	1
1996-02-28	1999-01-28	M4_MONTH	1
1999-01-15	1999-01-10	M4_DAY	5
1999-01-15	1998-12-31 22:00:00	M4_HOUR	12
1999-09-21 (Tuesday, la semana empieza en lunes)	1999-09-19	M4_COMPLETE_WEEK	1
1999-09-21	1999-09-19	M4_WEEK	0

Biblioteca básica de funciones de nivel 1

ExecuteGroup (Nombre de grupo)

Esta función ejecuta un grupo de conceptos en el orden lógico en el que deben hacerlo. Cuando se define un elemento, se le puede asignar un grupo determinado con un identificador único. En un grupo no deben existir bucles, ya que la función *ExecuteGroup ()* no acabaría nunca su ejecución.

EJEMPLO DE SINTAXIS

```
ExecuteGroup (
   VALUE VARIABLE STRING GroupName
   )
```

Flatten (Nodo destino)

Esta función es de ámbito registro. Se utiliza cuando se dispone de un registro con una serie de elementos, que a su vez tienen tramos. Esta función convierte ese registro en un conjunto de ellos y elimina los tramos, ya que la base de datos no permite su almacenamiento.

En ocasiones se necesita consultar la información recogida en los tramos, que podríamos denominar 3D a modo de ejemplo.

Se puede decir que la función *Flatten* convierte datos de 3D a 2D para tener acceso al contenido de los tramos. Entendemos por 2D un conjunto de datos expuestos en forma de tabla con filas y columnas. Esta conversión se hace sin pérdida de información.

Existe la posibilidad de volver al estado original después de ejecutar la función *Flatten*. Esto se llevaría a cabo mediante la función *Unflatten (elemento...[opcional])*.

EJEMPLO DE SINTAXIS

Si no se especifica el nodo destino, se trabajará con el mismo nodo.

```
Flatten (
   [VALUE VARIABLE STRING DestinationNode]
```

```
)
```

Unflatten (elemento...[opcional])

Esta función es de ámbito bloque. Su forma de actuación consiste en tomar un conjunto de registros y comprobar si tienen determinada estructura, de modo que si uno de los elementos tiene el tipo interno *StartDate* y otro elemento tiene el tipo interno *EndDate*, junta el conjunto de registros en uno solo que disponga de tramos.

Las fechas de inicio y de fin son las que van a determinar las fechas de inicio y fin del tramo. Es importante tener en cuenta que no puede haber solapamiento de fechas.

Se puede indicar a esta función que agrupe un conjunto de registros utilizando como criterio un determinado elemento.

Por ejemplo, si de un conjunto formado por veinte registros se tienen diez de ellos con un valor de elemento y el resto con otro valor de elemento, crearía dos registros distintos con tramos, uno para cada grupo de elementos.

Si no se tiene activado el modo tramos y no se tiene más de un registro, la función no actuaría. Si, por el contrario, el modo Tramos está activo y sólo hay un registro, se crearía un registro con tramos.

EJEMPLO DE SINTAXIS

```
Unflatten (
   [VALUE VARIABLE STRING Item]
)
```

JoinSlices ( )

Esta función permite enlazar todos los registros de un bloque activo, creando así un nuevo registro y borrando todos los originales. Todos los elementos del nuevo registro tendrán un tramo para cada periodo de intersección.

El resultado de los valores del elemento será la suma de los registros si el tipo es numérico, o el último registro si el tipo es cadena.

Sort ([índice])

A los nodos se les puede asignar una serie de índices que permitan clasificarlos de forma ascendente o descendente, tomando como referencia uno o varios elementos. Si no se indica un tipo de índice, se utilizará el que esté definido por defecto (sólo uno podrá estar marcado para que actúe por defecto).

La función Sort altera físicamente en memoria el orden de los registros, a diferencia del AddSortFilter, que sólo actúa a nivel de navegación.

EJEMPLO DE SINTAXIS

```
Sort (
   [VALUE NUMBER Index]
)
```

BSearch (Índice, Valores...)

Esta función realiza búsquedas binarias en el bloque activo (conjunto de registros), utilizando para ello uno o varios criterios de búsqueda.

Es necesario que el conjunto de registros esté ordenado. Si no es así, la función lo ordena.

Los argumentos correspondientes a esta función son:

- Índice

  Corresponde al identificador del índice. Ya en su descripción se indica el tipo de ordenación utilizado.

- Valores

  Contiene los elementos que forman parte del índice. Sólo se indican aquellos valores que se deben tener en cuenta para llevar a cabo la búsqueda.

Sólo se puede buscar por igualdad, excepto con el último elemento que puede ser igual, mayor, menor, etc., ya que si no fuera así, no podrían realizarse búsquedas binarias.

EJEMPLO DE SINTAXIS

```
BSearch (
```

```
    VALUE NUMBER Index, VALUE VARIANT Value
)
```

DumpRegister ( )

La función *DumpRegister ( )* se utiliza únicamente con fines de depuración. Esta función no recibe ningún argumento y graba en un archivo de traza, *C:\TEMP\VMREG.LOG*, el registro del nodo que está activo cuando se llama a la función.

El archivo de traza se guarda en el subdirectorio temporal del puesto de trabajo cliente.

La función *DumpRegister ( )* no puede utilizarse para grabar registros a un archivo distinto del archivo de traza predeterminado.

DumpNode ( )

Esta función graba en un archivo todos los registros correspondientes a los bloques que contiene el nodo activo.

Esta función se ejecuta con fines de depuración.

DumpChannel ( )

Esta función graba en un archivo información del Meta4Object activo.

Esta función se ejecuta con fines de depuración.

DumpStatistics ( )

Esta función graba en un archivo información resumen, tal como el número total de registros, la memoria utilizada, etc.

AppStartDate ( )

Esta función devuelve el resultado de la ejecución de la fecha de inicio. Su funcionamiento es similar al de la función *GetStartDate*.

AppEndDate ( )

Esta función devuelve el resultado de la ejecución de la fecha de fin. Su funcionamiento es similar al de la función *GetEndDate*.

AppRunDate ( )

Esta función devuelve la fecha de ejecución. Su funcionamiento es similar al de la función *GetRunDate*.

SetInserted ( )

Esta función marca un registro como nuevo. Se puede utilizar cuando se necesiten copiar registros de una tabla a otra.

Los registros de la tabla origen se marcan como insertados, y no se copiarán en la tabla destino hasta el momento en que se lleve a cabo la función *Persist* ( ).

Sólo es posible asignar una marca cuando se va a insertar un nuevo registro. El ámbito de aplicación de esta función es sólo para registros que no dispongan de marcas.

GetSliceMode ( )

Esta función devuelve el modo de ejecución con tramos para llevar a cabo su cálculo.

EJEMPLO DE SINTAXIS

```
GetSliceMode (
   VALUE NUMBER SliceMode
)
```

SetSliceMode (ModoTramo)

Esta función asigna al modo de ejecución con tramos el valor *True* o *False*, necesario para realizar cálculos posteriores sobre él.

EJEMPLO DE SINTAXIS

```
SetSliceMode (
   VALUE NUMBER SliceMode
)
```

GetArgumentNumber ( )

Esta función devuelve el número de argumentos que se han asignado a un elemento. Puede constar de una parte fija, de una parte variable, o de ambas.

GetArgument (Arg_pos)

Esta función devuelve el valor del argumento cuya posición se ha indicado como argumento. La posición debe ser un valor comprendido entre 1 y n.

EJEMPLO DE SINTAXIS

```
GetArgument (
   VALUE NUMBER ArgPosition
)
```

Funciones para manejo de registros

Las funciones recogidas en este apartado permiten realizar diversas operaciones sobre registros pertenecientes a una estructura de nodos.

La tabla siguiente recoge todas las funciones pertenecientes a esta categoría:

AddRegister ( )	SetValue (Nodo, Elemento, Registro, [Tramo], Valor)	IsDeleted ( )	SetFilterArgument (ID_Filtro, ID_Arg, Arg_Val, ...)
InsertRegister ( )	SetValuePriority (Prioridad, Nodo, Elemento, Registro, [Tramo], Valor)	IsNew ( )	CopyFiltersFrom (Meta4Object, Nodo)
DeleteRegister ( )	SetPriority (Prioridad, Nodo, Elemento, Registro, [Tramo])	IsUpdated ( )	ChannelAttribGetValue (Instancia de Meta4Object, Nodo, Elemento, Atributo)
DeleteAllRegisters ( )	GetValue (Nodo, Elemento, Registro, [Tramo])	IsUpdatedDB ( )	ChannelAttribCall (Argumentos,..., Instancia de Meta4Object, Nodo, Elemento, Atributo)
DestroyRegister ( )	SetChannelValue (Instancia de Meta4Object, Nodo, Elemento, Registro, [Tramo], Valor)	CloneRegister	AddSortFilter ( )
DestroyAllRegisters ( )	GetChannelValue (Instancia de Meta4Object, Nodo, Elemento, Registro, [Tramo], Valor)	CloneRegister	SendRegisterMark (Modo de envío)
DestroyBlock ( )	ChannelCall (Argumentos,..., Meta4Object, Instancia, Nodo, Elemento)	CopyRegister (Meta4Object, Nodo, [ÍndiceRegistro], [Filtro del elemento origen], [Filtro del elemento destino])	SendBlockMark (Modo de envío)

Begin ( )	FindRegister (Elemento1, Valor1, Criterio1, …, Elementon, Valorn, Criterion, [ÍndiceRegistro])	AddFilter (Código/ operación, [Estático], [Nombre del argumento_i], [Valor del argumento_i]...)	SendNodeMark (Modo de envío)
End ( )	IsEOF ( )	RemoveFilter ( )	ExecuteBranch (Ámbito, Nombre del elemento)
MoveToEOF ( )	GetCurrent ( )	CleanFilter ( )	
MoveTo (ÍndiceRegistro)	GoPrevious ( )	RefreshFilter ( )	
Count ( )	Gonext ( )	GetFilterArgument (ID_Filtro, ID_Argumento)	

AddRegister ( )

La función *AddRegister ( )* añade un nuevo registro al nodo en el que se ejecuta, y se sitúa en él. No recibe argumentos.

También permite añadir un registro en un nodo distinto del actual, siempre y cuando se encuentre en el mismo Meta4Object. Para ello, se incluirá el nombre de la estructura de nodo a la que se quiere añadir el registro en el nombre de la función, como por ejemplo:

```
ID_EN.AddRegister ( )
```

A continuación, será necesario asignar valores a los elementos de tipo campo del nuevo registro, que hasta entonces mostrará los valores asignados por defecto.

Para asignar valor a estos elementos, es necesario escribir el nombre del elemento seguido del signo igual "=" y a continuación el valor que se desee asignar. También se puede añadir un registro llamando a un elemento o creando un registro vacío, entre otras posibilidades.

Si se ejecuta la función *AddRegister ( )* sobre el nodo activo, el puntero que recorre la estructura de nodos se posiciona en el nuevo registro, que será el último registro del nodo.

Si crea un nuevo registro con AddRegister() y le asigna valores, éste permanece en memoria hasta que decida almacenarlo en la base de datos. En este caso, los cambios no se harán efectivos hasta que no ejecute la función Persist_tree ( ).

Si se ejecuta la función *AddRegister ( )* sobre otro nodo perteneciente al mismo Meta4Object, el puntero se sitúa en el nodo donde se añade el registro.

Los valores de los elementos conectados se rellenan automáticamente al añadir el registro, pero sólo se dará este comportamiento si se han conectado los elementos de los nodos padre e hijo entre los que existe una relación de clave externa.

Esto se lleva a cabo en este u otro nodo, dependiendo de dónde se encuentre el puntero en el momento en el que se cree el nuevo registro.

InsertRegister ( )

Esta función inserta un nuevo registro en el nodo actual. A diferencia de *AddRegister ( )*, que añade un registro en blanco al final del nodo, *InsertRegister ( )* crea el registro en la posición anterior a la que se encuentra situado el puntero. No recibe argumentos.

Para añadir un nuevo registro a un nodo distinto del actual, es necesario incluir el nombre de la estructura de nodo con la que se quiere trabajar en el nombre de la función. Por ejemplo:

```
ID_EN.InsertRegister ( )
```

Tras crear el registro vacío, se pueden asignar valores a los distintos elementos de tipo campo del nuevo registro. Para asignar valor a un elemento, es suficiente con escribir el nombre del elemento seguido del signo igual y del valor que se le quiere asignar. Si el nuevo registro se ha creado en un nodo distinto del actual, el nombre de los campos debe ir precedido por el nombre del nodo y por un punto. Por ejemplo:

```
ID_EN.elemento = "Valor"
```

Si se ejecuta la función *InsertRegister ( )* sobre el nodo activo, el puntero que recorre la estructura de nodos se posicionará en el nuevo registro.

Recuerde que el nuevo registro permanece en memoria, y que los cambios no se harán efectivos en la base de datos hasta que no llame a la función Persist_tree ( ).

Si se ejecuta la función *InsertRegister ( )* sobre otro nodo del mismo

Meta4Object, el puntero permanecerá en la misma posición en la que se encontraba antes de llamar a la función.

EJEMPLO

El siguiente código duplica todos los registros de la estructura de nodos en la que está posicionado el puntero.

El campo que constituye la clave primaria recoge un valor numérico. El código LN4 rellenará automáticamente el valor de este campo con el valor obtenido al elevar al cuadrado la clave primaria del registro que duplica.

Si ya existe un registro con este valor calculado en su clave primaria, se elevará al cuadrado, y así sucesivamente hasta encontrar un valor único.

```
Exponente = 2
Begin ( )
Do
For n= 0 to ( NumeroRegistro - 1 )
   If Id_Empleado = 1 Then
      IdEmp = 3
   Else
      IdEmp = Power ( Id_Empleado, Exponente )
   EndIf
NomEmp = N_Empleado
   EdadEmp = Edad_Empleado
   HayOtro = FindRegister ( "Id_Empleado", IdEmp, EQUAL )
   While HayOtro <> -1
      Exponente = Exponente + 1
      IdEmp = Power (IdEmp, Exponente )
      HayOtro = FindRegister ( "Id_Empleado", IdEmp, EQUAL )
   WEnd
   Exponente = 2
InsertRegister ( )
   Id_Empleado = IdEmp
   N_Empleado = NomEmp
   Edad_Empleado = EdadEmp
   Gonext ( )
Until IsEOF ( )=M4_TRUE
```

DeleteRegister ( )

Esta función no recibe argumentos y permite marcar como borrado el registro del nodo activo sobre el que está posicionado el puntero.

Para posicionarse en un registro específico del nodo activo, es necesario utilizar la función *MoveTo (ÍndiceRegistro).*

La función *DeleteRegister ( )* también puede utilizarse para eliminar un registro de otro nodo correspondiente al mismo u otro Meta4Object. Es suficiente con incluir en el nombre de la función el nombre de la estructura de nodo sobre la que se quieren realizar los cambios.

Si quiere eliminar un registro de un nodo distinto al nodo activo, es necesario posicionarse en el registro que se quiere eliminar.

DeleteAllRegisters ( )

Esta función permite marcar como borrados todos los registros de un bloque. No recibe argumentos.

Debe recordar que no se harán efectivos los cambios en la base de datos física hasta que no llame a la función *Persist_tree ( )*.

DestroyRegister ( )

Esta función permite eliminar el registro activo en el que está posicionado el puntero. No recibe argumentos, y después de que se haya ejecutado, el registro siguiente al que ha sido borrado pasa a ser el registro activo.

En caso de que el registro eliminado sea el último de un bloque, no quedará ningun registro como activo.

EJEMPLO

La siguiente tabla muestra el estado de una serie de registros antes de ejecutar la función DestroyRegister.

Registro 1	Registro inactivo	
Registro 2	Registro activo	Registro a eliminar
Registro 3	Registro inactivo	

La siguiente tabla muestra el estado de los registros anteriores después de ejecutar la función DestroyRegister.

Registro 1	Registro inactivo	
Registro 3	Registro activo	

DestroyAllRegisters ( )

Esta función permite eliminar todos los registros del bloque seleccionado. No recibe argumentos.

DestroyBlock ( )

Esta función permite eliminar el bloque activo, así como los elementos de bloque que existan. No recibe argumentos.

Begin ( )

Esta función permite posicionar el puntero en el primer registro del nodo activo. No recibe argumentos, y es equivalente a la función *MoveTo (0).*

También permite situarse en el primer registro de otro nodo del mismo Meta4Object. Para ello, es suficiente con incluir en el nombre de la función el de la estructura de nodo con la que se quiere trabajar.

Por ejemplo, para eliminar el primer registro del nodo *NombreNodo* desde el nodo llamado NODO2, se ejecutarían las siguientes órdenes:

```
ID_EN.Begin ( )
ID_EN.DeleteRegister ( )
```

End ( )

Esta función permite mover el registro activo a la última posición del conjunto seleccionado. No recibe argumentos.

MoveToEOF ( )

Esta función permite mover el puntero del nodo activo a la posición EOF. No debe confundirse la variable lógica EOF con el registro que ocupa la última posición en el nodo. La función no recibe argumentos.

Es posible mover el puntero a la posición EOF de otro nodo del mismo Meta4Object. Esta acción se puede llevar a cabo para cualquier elemento del sistema, tan sólo con incluir en el nombre de la función el nombre de la estructura de nodo con la que se quiere trabajar.

EJEMPLO

Si está situado en el nodo NODO1 y quiere situarse en la posición EOF del nodo NODO2, debería ejecutar la función del siguiente modo.

```
TINodo2.MoveToEOF ( )
```

MoveTo (ÍndiceRegistro)

Esta función mueve el puntero del nodo activo al registro situado en la posición ***n*** a la que se desee acceder. La función recibe como argumento el índice del registro al que se debe mover el puntero.

Los registros de un nodo se numeran comenzando por el número 0. De esta forma, si se quiere mover al primer registro del nodo activo, se ejecuta la orden *MoveTo(0);* para moverse al registro que ocupa la segunda posición, se ejecuta la orden *MoveTo(1)*, y así sucesivamente.

La función *MoveTo(n)* permite situar el puntero en un registro específico con diferentes propósitos, entre otros:

- Marcar para borrar el registro sobre el que está posicionado el puntero mediante la función *DeleteRegister ( ).*

  Este registro se borrará físicamente cuando se realice la próxima transacción.
- Insertar un nuevo registro en blanco a continuación del registro sobre el que está situado el puntero. Se utilizará para ello la función *InsertRegister ( ).*
- Ver el valor de un elemento de tipo campo o ejecutar un elemento de tipo método o concepto definido en el registro sobre el que se sitúa el puntero.

  Esta funcionalidad sólo se empleará si se quiere utilizar un elemento del nodo activo o de cualquier otro nodo.

EJEMPLO

Este ejemplo recorre todos los registros correspondientes al nodo activo, lee los valores del elemento NOMBRE y muestra este valor en una ventana de diálogo.

```
n=0
MoveTo(n)
Do
   MessageBox ("Aviso", NOMBRE)
   n=n+1
   MoveTo ( n )
Until IsEOF ( )=M4_TRUE
```

LN4 incluye otras dos funciones para moverse entre los registros de un nodo de forma secuencial: Gonext ( ) y GoPrevious ( ).

Count ( )

Esta función no recibe argumentos y devuelve el número de registros que están incluidos en el bloque activo.

También se puede aplicar la función *Count ( )* para saber cuántos registros incluye otro nodo del mismo Meta4Object. Será suficiente con incluir en el nombre de la función el nombre de la estructura de nodos cuyos registros se quiere contabilizar. Por ejemplo:

```
NoRegistros=ID_EN.Count ( )
```

En este caso, la variable **NoRegistros** recogerá el número de registros que incluye el nodo NombreNodo.

Puede utilizarse esta función junto a una de las sentencias que se exponen en el apartado *Estructuras de control*.

EJEMPLO

Suponga que tiene una estructura de nodo llamada HORAS_EXTRAS. El nodo HORAS_REALIZADAS de esta estructura recoge las horas extras realizadas por los empleados en el último mes.

Para calcular el número total de horas extras realizadas por todos los empleados, se procedería de la siguiente forma:

```
TotalHorasExtras = 0
NoHoras=HORAS_EXTRAS.Count ( )
For n=0 To NoHoras-1
   TotalHorasExtras=TotalHorasExtras + HORAS_EXTRAS.HORAS_EXTRAS
```

```
Next
MessageBox (“Aviso”, “Se han totalizado “ + TotalHorasExtras + “
horas extras”)
```

SetValue (Nodo, Elemento, Registro, [Tramo], Valor)

Esta función permite asignar un valor a un nodo, un elemento o un registro, o a uno de sus tramos.

La función *SetValue ( )* recibirá como argumentos el nombre del nodo, el nombre del elemento, el nombre del registro y el valor que se quiere asignar al elemento. El argumento *[Tramo]* corresponde al nombre del tramo, y es un argumento opcional, ya que si no se quiere asignar un valor a un tramo, se omite.

Únicamente se utilizará la función *SetValue ( )* cuando se desee asignar un valor a un elemento de un nodo cuyos nombres se reciben como variables o elementos.

En este último caso, conocemos el elemento en tiempo de ejecución, pero no en tiempo de diseño. Es decir, se debe utilizar *SetValue ( )* cuando se tiene que hacer una asignación de valor a un elemento cuyo nombre, nodo o tramo se desconoce.

Si se conoce el nombre del elemento y el nodo en el que se encuentra en tiempo de diseño, no se debe utilizar la función *SetValue ( )*. En su lugar, se le debe asignar un valor directamente, mediante una asignación del siguiente tipo:

```
Nodo.elemento = Valor
```

Ejemplo

El siguiente código realiza las siguientes acciones:

- Se indica a qué grupo de usuarios pertenece el usuario SUPERV.
- A continuación, se busca en la estructura de nodo PERMISOS_CONCEPTO_GRUPO las tablas de valor y los conceptos sobre los que tiene permisos el grupo de usuarios al que pertenece el usuario SUPERV.

Sean, por ejemplo, los siguientes registros:

Tabla	Concepto	Grupo
VAL_CATEGORIA	600	ADMIN
VAL_CONVENIO	560	ADMIN

El primer registro indica que los usuarios incluidos en el grupo ADMIN pueden modificar el valor del concepto 600 en la tabla de valor VAL_CATEGORIA.

El segundo registro indica que los usuarios incluidos en el grupo ADMIN pueden modificar el valor del concepto 560 en la tabla de valor VAL_CONVENIO.

- Tras consultar esta estructura de nodo, el método busca en la estructura de nodo **AcumuladoLargo** el valor que se ha asignado en la última paga calculada a los conceptos sobre los que tenga permiso el usuario SUPERV en la tabla PERMISOS_VALORES_CONCEPTO.
- Se supone que los registros del nodo **AcumuladoLargo** están ordenados de menor a mayor según los valores del campo FECHA_PAGO.
- Finalmente, en las tablas de valor sobre las que tenga permiso el usuario, se graban los valores recuperados del acumulado.

En este ejemplo se utilizarán las siguientes estructuras de nodo:

Grupos_Usuarios

Grupo	Usuario
ADMIN	M25
ADMIN	M25
SYSTEM	M25

Permisos_Valores_Concepto

Tabla_Valor	Concepto	Grupo
VAL_CATEGORIA	600	ADMIN
VAL_CONVENIO	560	ADMIN
VAL_CONVENIO	670	SYSTEM
VAL_CONVENIO	900	SYSTEM

Acumulado_Largo

ID_EMPLEADO	FEC_IMPU	FEC_PAGO	600	560	…….	1002
01	25-01-97	25-01-97	13000	2500	.....	90000
02	25-01-97	25-01-97	15000	6700	.....	98000
....	.....	.....	.....	.....	.....	.....
01	25-02-98	25-02-98	12000	3000	.....	100000
02	25-02-98	25-02-98	34000	5600	.....	120000

El método se ejecuta desde el nodo asociado a la estructura de nodo Permisos_Valores_ Concepto.

EJEMPLO

```
strGrupoUsuario = "ADMIN"
intRegistro = FindRegister ( "GRUPO", strGrupoUsuario, EQUAL)
If intRegistro = -1 Then
     MessageBox ( "Meta4", "No existe ningún permiso para este grupo
     de usuario")
Else
   intBuscoaPartirDe=intRegistro
   MoveTo (intRegistro)
   strTablaValor = TABLA_VALOR
   strConcepto = CONCEPTO
   Acumulado.End ( )
   dtUltimaPaga=Acumulado.FEC_PAGO
intRegAcum=Acumulado.FindRegister ("ID_EMPLEADO", "01", EQUAL,
"FEC_IMPU", dtUltimaPaga, EQUAL, "FEC_PAGO", dtUltimaPaga, EQUAL)
   If intRegAcum = -1 Then
      MessageBox("Meta4", "No se ha calculado este empleado en la
      última paga")
   Else
      Acumulado.MoveTo (intRegAcum)
GetValue ("ACUMULADO", strConcepto, -1, intValorConcepto)
     Call ( strTablaValor, "InsertRegister" )
SetValue ( strTablaValor, "Concepto", strConcepto )
     SetValue ( strTablaValor, "Valor", intValorConcepto )
     SetValue ( strTablaValor, "Fec_ini", Today ( ))
     SetValue ( strTablaValor, "Fec_fin", NullValue ( ) )
   EndIf
Do
intRegistro = FindRegister ( "GRUPO", strGrupoUsuario, EQUAL,
intBuscoaPartirDe)
     intBuscoaPartirDe=intRegistro
     MoveTo (intRegistro)
     strTablaValor = TABLA_VALOR
     strConcepto = CONCEPTO
     Acumulado.End ( )
     dtUltimaPaga=Acumulado.FEC_IMPU
intRegAcum=Acumulado.FindRegister ("ID_EMPLEADO", "01", EQUAL,
"FEC_IMPU", dtUltimaPaga, EQUAL, "FEC_PAGO", dtUltimaPaga, EQUAL)
     If intRegAcum = -1 Then
```

```
        MessageBox("Meta4", "No se ha calculado el empleado en la última
        paga")
        Else
        Acumulado.MoveTo (intRegAcum)
    GetValue ("ACUMULADO", strConcepto, intValorConcepto)
              Call ( strTablaValor, "InsertRegister" )
    SetValue ( strTablaValor, "Concepto", strConcepto )
              SetValue ( strTablaValor, "Valor", intValorConcepto )
              SetValue ( strTablaValor, "Fec_ini", Today ( ))
              SetValue ( strTablaValor, "Fec_fin", NullValue ( ) )
    EndIf
    Until IsEOF ( )=M4TRUE OR IntRegister = -1
    EndIf
```

Call (Argumentos,..., Nodo, Elemento)

La función *Call ( )* se utiliza para llamar a otros elementos de forma indirecta. Sólo se debe utilizar cuando el método que hace la llamada no conozca el nombre del método que se va a ejecutar en tiempo de diseño, es decir, cuando el nombre de la función se obtenga como resultado de la ejecución de otro método.

En caso de que se conozca el nombre de la función o método al que se quiere llamar en tiempo de diseño, las llamadas se harán de manera directa, escribiendo su nombre, proceso que resulta bastante rápido y más eficiente.

La función *Call ( )* recibe como argumentos el nombre del método o concepto que se quiere ejecutar, con todos sus argumentos. Con la función ***Call*** ( ) sólo se pueden hacer llamadas de tipo método o concepto.

La sintaxis de la función *Call ( )* depende de si se llama a un método del mismo nodo, o de otro nodo del mismo Meta4Object.

Si se utiliza *Call ( )* para llamar a un método del nodo desde el que se hace la llamada, en la lista de argumentos se escribirán entre paréntesis todos los argumentos que necesite la función a la que se llama, separados por comas.

```
Call  ( argumento1, argumento2, …, argumenton, nodo, elemento )
```

Si se utiliza *Call ( )* para llamar a un método de otro nodo del mismo Meta4Object. En la lista de argumentos se escribirá entre paréntesis el nombre del nodo, los argumentos del método (si los hay) y el nombre del método.

EJEMPLO

Para llamar al método *InsertRegister ( )* del nodo EMPLEADOS, se debe escribir:

```
Call ( "EMPLEADOS", "InsertRegister")
```

SetValuePriority (Prioridad, Nodo, Elemento, Registro, [Tramo], Valor)

Esta función asigna un valor indicado como argumento al nombre del elemento, asignándole una determinada prioridad entre 0 y 3 (con 0 para nivel de prioridad máximo y 3 para nivel de prioridad mínimo).

Sus argumentos son la prioridad que tiene asignada un elemento, los nombres del nodo, elemento, registro y tramo al que se hace referencia, y el valor que se pretende asignar al elemento.

La prioridad no sólo se asigna a elementos, sino también a tramos. Sólo es posible compactar dos elementos o tramos si ambos tienen las mismas prioridades.

El argumento del tramo es opcional. Si el argumento *Registro* es -1, se utiliza el tramo activo. Si en la ejecución con tramos el argumento correspondiente a tramos no se especifica, se utilizaría el del tramo activo por defecto.

Cuando con esta función se intenta sobrescribir el valor de un elemento se comprueba el valor asignado al argumento ***Prioridad,*** momento en el cual pueden suceder dos cosas:

- Si la prioridad del elemento es menor o igual que aquélla con la que se compara, se sobrescribe el valor del elemento con el valor que se pasa como argumento.
- Si la prioridad del elemento es mayor que aquélla con la que se compara, no se sobrescribe el valor del elemento.

EJEMPLO DE SINTAXIS

```
SetValuePriority (
   VALUE NUMBER Priority, VALUE VARIABLE STRING Node, VALUE
   VARIABLE STRING Item, VALUE NUMBER Record, [VALUE NUMBER Slice]
)
```

SetPriority (Prioridad, Nodo, Elemento, Registro, [Tramo])

Esta función cambia directamente el valor correspondiente a la prioridad de un elemento. A diferencia de la función SetValuePriority, ésta no modifica el valor del elemento.

Sus argumentos son la prioridad que se va a asignar al elemento, y los nombres de nodo, elemento, registro y tramo a los que se hace referencia. El argumento de Tramo es opcional. Si se ejecuta la función con tramos y no se especifica su argumento correspondiente, se usa el tramo activo.

EJEMPLO DE SINTAXIS

```
SetPriority (
   VALUE NUMBER Priority, VALUE VARIABLE STRING Node, VALUE
   VARIABLE STRING Item, VALUE NUMBER Record, [VALUE NUMBER Slice]
)
```

GetValue (Nodo, Elemento, Registro, [Tramo])

Esta función recoge el valor de un campo o de un tramo de un campo, así como del nodo y del registro que aparecen como argumentos. Dichos argumentos son variables o elementos que contienen el nombre del nodo, el nombre del elemento y el índice del registro.

El argumento Tramo es opcional, e identifica el tramo cuyo valor se quiere recuperar. Si se ejecuta con tramos y no se especifica el argumento correspondiente, se usa el tramo activo.

EJEMPLO DE SINTAXIS

```
GetValue (
   VALUE VARIABLE STRING Node, VALUE VARIABLE STRING Item, VALUE
   NUMBER Record, VALUE NUMBER Slice
)
```

Únicamente se debe utilizar la función de indirección *GetValue ( )* cuando se desee obtener el valor de un elemento o de un tramo cuyo nombre o nodo se reciba a través de una variable que se desconoce a la hora de escribir el código.

Si se conoce el nombre del elemento, siempre será más fácil obtener su valor

directamente mediante una asignación del siguiente tipo:

```
Variable = ID_EN.elemento
```

Para obtener el valor de un tramo, se debe indicar (aun siendo opcional) el tramo en el argumento que recibe la función *GetValue ( )* en cuarto lugar.

SetChannelValue (Instancia de Meta4Object, Nodo, Elemento, Registro, [Tramo], Valor)

La diferencia de esta función con respecto a SetValue estriba en el ámbito de su actuación:

- *SetValue*: hace referencia a un elemento del mismo Meta4Object que el que se encuentra activo.
- *SetChannelValue*: hace referencia a un elemento de un Meta4Object distinto al que se encuentra activo, aunque pudiera ser también el mismo Meta4Object. Para tener acceso a un Meta4Object distinto al activo, se han de realizar llamadas de nivel 2.

Esta función recibe como argumentos una instancia de un Meta4Object, los nombres del nodo, elemento y registro a los que se hace referencia, el tramo que va a considerar la función, y el valor que se va a asignar a un elemento dado en una instancia

El argumento *Tramo* es un campo opcional. Si la función se ejecuta con tramos y no se especifica el argumento correspondiente, se usa el tramo activo.

EJEMPLO DE SINTAXIS

```
SetChannelValue (
   VALUE VARIABLE STRING Meta4Object, VALUE VARIABLE STRING Node,
   VALUE VARIABLE STRING Item, VALUE NUMBER Record, [VALUE NUMBER
   Slice,] VALUE VARIANT Value
)
```

GetChannelValue (Instancia de Meta4Object, Nodo, Elemento, Registro, [Tramo], Valor)

Esta función lee el valor de un elemento dado en una instancia. La diferencia con la función GetValue estriba en el ámbito de actuación de la misma:

- *GetValue*: hace referencia a un elemento del mismo Meta4Object que el activo.

- *GetChannelValue*: hace referencia a un elemento de un Meta4Object distinto al activo, aunque pudiera ser también el mismo.

  Para tener acceso a un Meta4Object distinto al activo, se han de realizar llamadas de nivel 2.

Los argumentos correspondientes a esta función son una instancia creada en un Meta4Object, los nombres de nodo, elemento y registro a los que se hace referencia, el tramo que se va a condiserar, y el valor que se va a asignar al elemento dado en la instancia dada.

El argumento *Tramo* es un campo opcional y corresponde al tramo utilizado que va a considerar la función GetChannelValue.

Al utilizar esta función con tramos, existen las siguientes posibilidades:

- Si esta función se ejecuta con tramos y no se especifica el argumento correspondiente, se usa el tramo activo.
- Si el argumento Registro tiene el valor -1, la función lee el registro activo.
- Si se ejecuta con tramos y no se especifica el argumento correspondiente, se utiliza el tramo activo.

EJEMPLO DE SINTAXIS

```
GetChannelValue (
   VALUE VARIABLE STRING Meta4Object, VALUE VARIABLE STRING Node,
   VALUE VARIABLE STRING Item, VALUE NUMBER Record, [VALUE NUMBER
   Slice]
)
```

ChannelCall (Argumentos,..., Meta4Object, Instancia, Nodo, Elemento)

Esta función ejecuta un elemento en un nodo de una instancia de Meta4Object que no es el activo. La diferencia de esta función con *Call* estriba en el ámbito de actuación:

- *Call:* hace referencia a un elemento del mismo Meta4Object que el que se encuentra activo.
- *ChannelCall*: hace referencia a un elemento de un Meta4Object distinto del que se encuentra activo, aunque pudiera ser también el mismo Meta4Object.

  Para tener acceso a un Meta4Object distinto al activo, se han de realizar llamadas de nivel 2.

Los argumentos correspondientes a esta función son los argumentos de la función que se pretende ejecutar, el nombre del Meta4Object al que se hace referencia, el nombre de la instancia a la que se hace referencia (correspondiente al Meta4Object indicado como argumento), y los nombres del nodo y elemento a los que se hace referencia.

EJEMPLO DE SINTAXIS

```
ChannelCall (
   [VALUE VARIANT Argument1,] VALUE VARIABLE STRING Meta4Object,
   VALUE VARIABLE STRING Node, VALUE VARIABLE STRING Item
)
```

FindRegister (Elemento$_1$, Valor$_1$, Criterio$_1$, …, Elemento$_n$, Valor$_n$, Criterio$_n$, [ÍndiceRegistro])

Esta función busca los registros de un bloque que cumplen una serie de criterios que se han indicado como argumentos.

Para cada condición o criterio se deben indicar tres argumentos, y es posible escribir un número ilimitado de condiciones.

Si se asignan a la función *FindRegister ( )* dos o más condiciones, únicamente se recuperarán los registros que cumplan todas ellas; las condiciones se unirán mediante el operador lógico AND.

Los argumentos se deben indicar en el siguiente orden:

- Como primer argumento se ha de indicar el nombre del campo sobre el que se quiere realizar la búsqueda.
- Como segundo argumento se debe indicar el valor que se quiere buscar en el campo indicado como primer argumento.
- En el tercer argumento se debe indicar el criterio que se quiere utilizar para comparar los valores recogidos por el campo con el valor que se introduce como segundo argumento. Los criterios disponibles son los que se muestran en la siguiente tabla.

Criterio	Descripción
EQUAL	Localiza un registro en el que la cadena que se busca sea igual a la que se indica.
DISTINCT	Localiza un registro en el que la cadena que se busca sea distinta a la que se indica.
GREATER	Localiza un registro en el que la cadena que se busca sea más grande que la que se indica.
SMALLER	Localiza un registro en el que la cadena que se busca sea más pequeña que la que se indica.
GREATER_OR_EQUAL	Localiza un registro en el que la cadena que se busca sea más grande o igual que la que se indica.
SMALLER_OR_EQUAL	Localiza un registro en el que la cadena que se busca sea más pequeña o igual que la indicada.
EQUAL_OR_NULL	Localiza un registro en el que la cadena que se busca sea nula o igual que la indicada.
DISTINCT_OR_NULL	Localiza un registro en el que la cadena que se busca sea nula o distinta que la indicada.
GREATER_OR_NULL	Localiza un registro en el que la cadena que se busca sea nula o mayor que la indicada.
SMALLER_OR_NULL	Localiza un registro en el que la cadena que se busca sea nula o menor que la indicada.
GREATER_OR_EQUAL_OR_NULL	Localiza un registro en el que la cadena que se busca sea nula, mayor o igual que la indicada.
SMALLER_OR_EQUAL_OR_NULL	Localiza un registro en el que la cadena que se busca sea nula, menor o igual que la indicada.
CASE_EQUAL	Localiza un registro en el que la cadena que se busca sea igual a la indicada, distinguiendo entre mayúsculas y minúsculas.
CASE_DISTINCT	Localiza un registro en el que la cadena que se busca sea distinta a la indicada, distinguiendo entre mayúsculas y minúsculas.
CASE_GREATER	Localiza un registro en el que la cadena que se busca sea mayor que la indicada, distinguiendo entre mayúsculas y minúsculas.
CASE_SMALLER	Localiza un registro en el que la cadena que se busca sea menor que la indicada, distinguiendo entre mayúsculas y minúsculas.
CASE_GREATER_OR_EQUAL	Localiza un registro en el que la cadena que se busca sea mayor o igual que la indicada, distinguiendo entre mayúsculas y minúsculas.

Criterio	Descripción
CASE_SMALLER_OR_EQUAL	Localiza un registro en el que la cadena que se busca sea menor o igual que la indicada, distinguiendo entre mayúsculas y minúsculas.
CASE_EQUAL_OR_NULL	Localiza un registro en el que la cadena que se busca sea nula o igual que la indicada, distinguiendo entre mayúsculas y minúsculas.
CASE_DISTINCT_OR_NULL	Localiza un registro en el que la cadena que se busca sea nula o distinta que la indicada, distinguiendo entre mayúsculas y minúsculas.
CASE_GREATER_OR_NULL	Localiza un registro en el que la cadena que se busca sea nula o mayor que la indicada, distinguiendo entre mayúsculas y minúsculas.
CASE_SMALLER_OR_NULL	Localiza un registro en el que la cadena que se busca sea nula o menor que la indicada, distinguiendo entre mayúsculas y minúsculas.
CASE_GREATER_OR_EQUAL_OR_NULL	Localiza un registro en el que la cadena que se busca sea nula, mayor o igual que la indicada, distinguiendo entre mayúsculas y minúsculas.
CASE_SMALLER_OR_EQUAL_OR_NULL	Localiza un registro en el que la cadena que se busca sea nula, menor o igual que la indicada, distinguiendo entre mayúsculas y minúsculas.
REGULAR_EXPRESSION	Permite establecer una serie de requisitos que debe cumplir el valor del elemento. Por ejemplo, que el valor de éste empiece por una determinada cadena o un determinado carácter, etc.
REGULAR_EXPRESSION_OR_NULL	Permite establecer una serie de requisitos que debe cumplir el valor del elemento, a menos que se trate de un valor nulo. Por ejemplo, que el valor del elemento empiece por una determinada cadena o un determinado carácter, etc.
CASE_REGULAR_EXPRESSION	Permite establecer una serie de requisitos que debe cumplir el valor del elemento, distinguiendo entre mayúsculas y minúsculas. Por ejemplo, que el valor del elemento empiece por una determinada cadena o un determinado carácter, etc.
CASE_REGULAR_EXPRESSION_OR_NULL	Permite establecer una serie de requisitos que debe cumplir el valor del elemento, a menos que se trate de un valor nulo, distinguiendo entre mayúsculas y minúsculas. Por ejemplo, que el valor del elemento empiece por una determinada cadena o un determinado caracter, etc.

EJEMPLO

En este ejemplo se utiliza el criterio EQUAL. Este criterio permite buscar un registro cuyo nombre sea igual a Laura.

Este criterio puede ser sustituido por cualquier otro de los indicados en la siguiente tabla.

```
FindRegister ("NOMBRE", "Laura", EQUAL)
```

La función *FindRegister ( )* puede recibir un argumento opcional. Este argumento se debe escribir en última posición, detrás de todas las condiciones, e indica el número de registro a partir del cual se quiere comenzar la búsqueda, entre 0 y n-1.

Si la estructura de nodo incluye al menos un registro que cumpla las condiciones, la función devuelve el índice del primer registro de la estructura de nodo que cumpla las condiciones indicadas.

Si la estructura de nodo no incluye ningún registro que cumpla las condiciones, la función devuelve el valor -1.

EJEMPLO

En el nodo ABSENTISMOS se quieren eliminar los registros correspondientes al empleado 01, con una fecha comprendida entre el 01-07-1999 y el 22-07-1999.

El nodo ABSENTISMOS recoge el código del empleado en el elemento ID_EMPLEADO, y las fechas de inicio y de fin de los periodos de absentismo en los elementos FEC_INI y FEC_FIN.

Para eliminar estos registros, es necesario realizar una búsqueda para saber cuál es su índice. Una vez que se conoce el índice, se puede ejecutar la función DeleteRegister ( ). A continuación se muestra el código LN4 correspondiente a esta función a modo de ejemplo.

```
Register=FindRegister ( "ID_EMPLEADO", "01", EQUAL, "FEC_INI",
{1999-07-01 }, GREATER_OR_EQUAL, "FEC_FIN", { 1999-07-22 },
SMALLER_OR_EQUAL)
If Register = -1 Then
   MessageBox ("Meta4", "No se ha encontrado ningún registro que
   cumpla las condiciones")
Else
   BuscoaPartirDe=Register
   MoveTo (Register)
   DeleteRegister ( )
   Do
```

```
        Register=FindRegister ( "ID_EMPLEADO", "01", EQUAL,
        "FEC_INI", {1999-07-01 }, GREATER_OR_EQUAL, "FEC_FIN", {
        1999-07-22 }, SMALLER_OR_EQUAL, BuscoaPartirDe )
        MoveTo (Register)
        DeleteRegister ( )
        BuscoaPartirDe=Register + 1
    Until IsEOF ( )=M4_TRUE OR Register = -1
EndIf
```

IsEOF ( )

La función *IsEOF ( )* permite saber si el puntero que recorre la estructura de nodo ha alcanzado la variable lógica EOF: En este caso hay dos posibilidades:

- Si el puntero está posicionado en un registro del nodo activo, *IsEOF ( )* devuelve el valor M4_FALSO.
- Si el puntero está posicionado en la variable lógica EOF, fuera de cualquier registro, la función devuelve el valor M4_TRUE.

La función *IsEOF ( )* se utiliza para recorrer los registros de un nodo, junto a las sentencias de control *While…Wend*, y *Do…Until*.

Además, se puede utilizar para recorrer los registros de nodos distintos al nodo activo. Para ello, basta con cualificar el nombre de la función con el nombre de la estructura de nodo con la que se quiere trabajar.

EJEMPLO

```
Total=0
Begin( )
Do
      Total=Total+SALARIO_BASE
      Gonext( )
Until IsEOF( )=M4_TRUE
Total=0
Begin( )
While NOT IsEOF( )=M4_TRUE
      Total=Total+SALARIO_BASE
      Gonext( )
Wend
```

GetCurrent ( )

Esta función devuelve el índice correspondiente al registro del nodo activo sobre el que está situado el puntero.

Los registros de un nodo se empiezan a contar a partir del número 0. Así, si un nodo incluye 10 registros, éstos se numerarán del 0 al 9.

EJEMPLO

```
Begin ( )          ‘ Posiciona el puntero en el primer registro
Registro = GetCurrent ( )   ‘ Asigna a la variable Registro el valor 0
MoveTo(9)          ‘ Sitúa  el puntero en el décimo registro
Registro= GetCurrent ( )    ‘ Asigna a la variable Registro el valor 9.
```

GoPrevious ( )

Esta función permite mover el puntero que recorre el nodo activo una posición hacia atrás a partir del registro que se encuentra activo en ese momento.

EJEMPLO

```
End ( )       ‘Sitúa el puntero en el último registro del nodo actual
Goprevious ( )      ‘Mueve el puntero hasta el penúltimo registro
```

Gonext ( )

Esta función desplaza el puntero que recorre el nodo activo una posición hacia adelante, a partir del registro que se encuentre activo en ese momento.

EJEMPLO

```
Begin ( )         ‘Sitúa el puntero en el primer registro.
Registro=GetCurrent ( )  ‘Asigna a la variable Registro el valor 0.
Gonext ( )        ‘Mueve el puntero una posición hacia adelante.
Registro=GetCurrent ( )  ‘Asigna a la variable Registro el valor 1.
```

IsDeleted ( )

Esta función devuelve el valor M4_TRUE si el registro activo del bloque está marcado como borrado.

IsNew ( )

Esta función devuelve el valor M4_TRUE si el registro activo del bloque está marcado como nuevo.

IsNewAndDeleted( )

Esta función devuelve el valor M4_TRUE si el registro activo del bloque está marcado como nuevo y como borrado.

La función no tiene argumentos y se aplica al registro sobre el que se esté trabajando.

IsUpdated ( )

Esta función devuelve el valor M4_TRUE si el registro activo del bloque está marcado como actualizado.

IsUpdatedDB ( )

Devuelve el valor M4_TRUE si el registro está marcado como que tiene información actualizada que se desea grabar en base de datos.

CloneRegister

Esta función crea un nuevo registro, realiza una copia exacta del registro activo, y se posiciona en el nuevo registro.

CopyRegister (Meta4Object, Nodo, [ÍndiceRegistro], [Filtro del elemento origen], [Filtro del elemento destino])

Esta función permite copiar en el registro activo, tomando como origen el registro especificado por el Meta4Object, Nodo, ÍndiceRegistro. Se pueden dar las siguientes situaciones:

- Si el valor del Meta4Object está vacío, se utiliza el Meta4Object en el que esté posicionado.
- Si el *IndiceRegistro* es -1 o no se especifica (sin filtros), se usa como activo el nodo ***origen***.
- Por defecto, si no se especifica un filtro, sólo se consideran los elementos del registro (todos ellos).

Los argumentos correspondientes a esta función son el nombre del Meta4Object que se va a utilizar, el nombre del nodo al que se hace referencia, el índice que permite localizar el registro (opcional), y los filtros de los nodos origen y destino.

Para más información sobre filtros, consulte el apartado Especificación del filtro. Un filtro básico puede ser del tipo SCOPE_REGISTER + TYPE_PROPERTY.

EJEMPLO DE SINTAXIS

```
CopyRegister (
   VALUE VARIABLE STRING Meta4Object, VALUE VARIABLE STRING Node,
   [VALUE NUMBER RegisterIndex,] [VALUE NUMBER SourceItemFIlter,]
   [VALUE VARIABLE STRING DestItemFilter]
)
```

AddFilter (Código/operación, [Estático], [Nombre del argumento_i], [Valor del argumento_i]...)

Esta función permite añadir un filtro al nodo cargado en memoria, con el primer argumento como filtro.

EJEMPLO DE SINTAXIS

```
AddFilter (
   VALUE VARIANT Code_Operation, [VALUE NUMBER Static,] [VALUE
   VARIABLE STRING ArgId1,] [VALUE VARIANT ArgVal1]
)
```

Con esta función se pueden dar las siguientes posibilidades:

- Si el primer argumento es una cadena, se asume un código LN4 para determinar los registros visibles.
- Si el primer argumento es una macro especificada, se utilizará una operación de filtros.

  Con esta macro se podrá mostrar una lista en la que se incluyan ciertos registros y se excluyan otros.
- Si el argumento Estático tiene como valor M4_TRUE, el filtro es estático; si por el contrario es falso (0), el filtro será dinámico. Por defecto es estático.
- Si el códico del filtro tiene argumentos (código LN4), éstos se pueden indicar como argumentos, así como sus valores iniciales, justamente después de los argumentos Código y Estático.

Esta función devuelve el ID Filtro que debe utilizarse para cambiar o tomar valores como argumentos.

EJEMPLO

Con este código sólo se ven los registros que han sido borrados:

```
AddFilter (OP_DELETED)
```

Pero con éste se ven todos los registros excepto los que han sido borrados:

```
AddFilter (-OP_DELETED)
```

RemoveFilter ( )

Esta función permite eliminar el último filtro cargado en memoria que se haya añadido a un nodo. El filtro es la capacidad de los Meta4Object que permite seleccionar en un nodo aquellos registros que cumplen una determinada condición.

CleanFilter ( )

Esta función permite eliminar todos aquellos filtros que se encuentren en memoria añadidos a un nodo.

RefreshFilter ( )

Esta función refresca un filtro estático que se encuentra en memoria añadido a un nodo. Se debe tener en cuenta que si se añaden registros y no se ejecuta la función RefreshFilter, los datos no se actualizarán hasta que no se ejecute esta función.

GetFilterArgument (ID_Filtro, ID_Argumento)

Esta función muestra el valor actual del argumento indicado. Dicho valor no se puede cambiar, puesto que la operación se realiza con la función SetFilterArgument.

Los argumentos correspondientes son el identificador del filtro utilizado, y el identificador del argumento utilizado cuyo valor se va a mostrar con el uso de esta función. Los argumentos se numeran a partir del 0, y el valor del argumento queda determinado por la posición del registro o su identificador.

EJEMPLO DE SINTAXIS

```
GetFilterArgument (
   VALUE VARIABLE STRING IdFilter, VALUE VARIANT IdArg
)
```

SetFilterArgument (ID_Filtro, ID_Arg, Arg_Val, ...)

Esta función permite cambiar el valor actual del argumento indicado. Para ello recibe como argumentos el identificador del filtro utilizado, el identificador del argumento utilizado cuyo valor se mostrará al utilizar esta función, y el valor del argumento que se va a asignar.

Los argumentos se numeran a partir del 0 y se pueden utilizar uno o varios; y el argumento puede ser el especificado por un identificador o posición.

EJEMPLO DE SINTAXIS

```
SetFilterArgument (
   VALUE VARIABLE STRING IdFilter, VALUE VARIANT IdArg0, VALUE
   VARIANT ArgValue0
)
```

CopyFiltersFrom (Meta4Object, Nodo)

Esta función copia los filtros de un nodo o un Meta4Object indicados como argumentos y los asigna en el nodo activo. El argumento *Meta4Object* recoge el nombre del MetaObject con el que se va a trabajar con esta función. Si este argumento tiene como valor " ", se refiere al Meta4Object activo. Por su parte,

el argumento *Nodo* recoge el nombre del nodo al que se va a hacer referencia.

ChannelAttribGetValue (Instancia de Meta4Object, Nodo, Elemento, Atributo)

Del mismo modo que existen funciones de indirección para leer y asignar valores a elementos, existen funciones de indirección para ejecutar o leer atributos asociados a elementos de sistema, como por ejemplo el atributo *SysStartDate*.

En este caso, la función ChannelAttribGetValue devuelve el atributo del elemento que se ha indicado por indirección.

Los argumentos de la función son el identificador de la instancia correspondiente al Meta4Object activo que se va a utilizar, los nombres del nodo y elemento a los que se va a hacer referencia, y el identificador de cada uno de los atributos que tiene el elemento que se va a utilizar.

Esta función asigna el primer argumento a la cadena vacía, indicando así una instancia del Meta4Object actual.

Para cada atributo de los que puede tener un elemento, existe una serie de constantes definidas cuyo valor variará dependiendo de cada caso, pero que habrá que indicar siempre en el argumento ***Atributo***.

Los únicos atributos que no se pueden ejecutar son aquéllos que tienen argumentos por referencia.

EJEMPLO DE SINTAXIS

```
ChannelAttribGetValue (
   VALUE VARIABLE STRING Meta4Object, VALUE VARIABLE STRING Node,
   VALUE VARIABLE STRING Item, VALUE NUMBER Attribute
)
```

ChannelAttribCall (Argumentos,..., Instancia de Meta4Object, Nodo, Elemento, Atributo)

Esta función ejecuta el atributo del elemento indicado como argumento por indirección con el resto de los argumentos. Si se desea leer el valor de un atributo o ejecutar uno por indirección refiriéndose al mismo Meta4Object, es suficiente indicar comillas.

Si por otro lado se quiere utilizar otro Meta4Object distinto al actual, habría que indicar el nombre del Meta4Object que contiene el atributo al que se hace referencia.

El parámetro *Argumentos* corresponde a los argumentos complementarios que se indican como argumentos y que la función debe tener en cuenta a la hora de su ejecución. El número de argumentos es ilimitado, por tanto, se pueden tener tantos como sean necesarios, dependiendo del atributo.

La *Instancia de Meta4Object* identifica la instancia correspondiente al Meta4Object activo que se va a utilizar.

Finalmente, los argumentos nodo, elemento y atributo identifican respectivamente el nodo y elemento a los que se va a hacer referencia, y a cada uno de los atributos que tiene el elemento que se va a utilizar.

EJEMPLO DE SINTAXIS

```
ChannelAttribCall (
   VALUE VARIANT Arguments, VALUE VARIABLE STRING Meta4Object,
   VALUE VARIABLE STRING Node, VALUE VARIABLE STRING Item, VALUE
   VARIABLE STRING Attribute
)
```

AddSortFilter ( )

Esta función permite añadir un filtro para ordenar todos los registros correspondientes a un nodo, de acuerdo con el criterio indicado por el usuario. Admite de dos formas de ejecución, con un índice definido o un número "n" de parejas de elementos y un modo de ordenación. :

- AddSortfilter (Índice)
- AddSortfilter (Item_name, Order [asc/desc]...n pairs)

EJEMPLO DE SINTAXIS

```
AddSortFilter (
   VALUE VARIANT Index_Item, [VALUE NUMBER Order]
)
```

SendRegisterMark (Modo de envío)

Esta función permite ver las constantes definidas para *Modo de envío*. Permite marcar el registro activo para una serialización avanzada. El valor por defecto es No_BRANCH.

SendBlockMark (Modo de envío)

Esta función permite ver las constantes definidas para *Modo de envío*. Permite marcar el bloque activo para una serialización avanzada. El valor por defecto es No_BRANCH.

SendNodeMark (Modo de envío)

Esta función permite ver las constantes definidas para *Modo de envío*. Permite marcar el nodo activo para una serialización avanzada. El valor por defecto es No_BRANCH.

ExecuteBranch (Ámbito, Nombre del elemento)

Si se dispone de un Meta4Object con estructura jerárquica, es posible ejecutar toda una rama del Meta4Object, es decir, un nodo y todos los elementos que dependen de él.

Los argumentos correspondientes a esta función son:

- Ámbito

  Representa el ámbito sobre el que se va a ejecutar la función ExecuteBranch.

  - Si *Ámbito* es registro, la función ejecuta el *Nombre del elemento* en todos los nodos que dependan del nodo raíz de un registro.
  - Si *Ámbito* es bloque, la función ejecuta el *Nombre del elemento* en todos los nodos que dependan del nodo raíz de un bloque.
  - Si *Ámbito* es nodo, la función ejecuta el *Nombre del elemento* en todos los nodos.

- Nombre del elemento

  Indica el nombre del elemento al que se va a hacer referencia.

IsBlockUpdated(ai_bRecursive)

Esta función comprueba si alguno de los elementos de ámbito bloque del bloque actual ha sido modificado. Si alguno ha sido modificado retorna M4_TRUE y si no hay modificaciones retorna M4_FALSE. Además, si el argumento de la función es M4_TRUE chequea si alguno de los elementos de todos los registros del bloque actual ha sido modificado.

Los argumentos correspondientes a esta función son los siguientes:

- **ai_bRecursive :** número booleano que indica si se deben comprobar los cambios en los elementos de ámbito registro del bloque

IsNodeUpdated(ai_bRecursive)

Esta función comprueba si alguno de los elementos de ámbito nodo del nodo actual ha sido modificado. Si alguno ha sido modificado retorna M4_TRUE y si no hay modificaciones retorna M4_FALSE. Si el argumento de la función es M4_TRUE además chequea si alguno de los elementos de ámbito bloque de todos los bloques del nodo actual ha sido modificado y si alguno de los elementos de todos los registros de todos los bloques del nodo actual ha sido modificado

Los argumentos correspondientes a esta función son los siguientes:

- **ai_bRecursive:** número booleano que indica si se deben comprobar los cambios en los elementos de ámbito bloque y registro del nodo.

Funciones para la base de datos lógica

Las funciones que pertenecen a esta categoría permiten realizar operaciones de diverso tipo en la base de datos lógica. La siguiente tabla recoge el conjunto de funciones englobadas en este apartado:

BeginTransaction ( )	BeginDBTransaction	Delete_Prg ( )	Persist_tree ( )
EndTransaction (Commit / Rollback / Ejecutar postvalidación)	EndDBTransaction (Commit / Rollback / Ejecutar postvalidación)	Update_blk ( )	SetAutoloadMode (Modo)
ExecuteSQL ( )	Load_blk ( )	Update_prg ( )	SetNodeAutoloadMode (Modo)
ExecuteDirectSQL (ID_Conexión, Máscara,...)	Load_prg ( )	Insert_blk ( )	GetAutoloadMode ( )
ExecuteRealSQL (ID_Conexión,...)	Delete_Blk ( )	Insert_prg ( )	GetNodeAutoloadMode (Modo)

Tipos de transacciones

Antes de comenzar la descripción de las funciones pertenecientes a este apartado, vamos a hacer una distinción previa entre los dos tipos de transacciones posibles:

- Por un lado, están las transacciones básicas de base de datos. Para trabajar con éstas, se dispone de las siguientes funciones:
  - BeginDBTransaction ( )
  - EndDBTransaction (Acometer/Vuelta atrás/Ejecutar postvalidación)
- Por otro, están las transacciones de Meta4Object Engine, que permiten trabajar con métodos en los que se desconoce a priori si van a existir fallos o no en su ejecución, como la función Persist(). Si los métodos fallan, se permite continuar ejecutando el código.

  Las funciones para trabajar con estas transacciones son:
  - BeginVMTransaction ( )
  - EndVMTransaction (Acometer/Vuelta atrás/Ejecutar postvalidación)

Además, existe otro conjunto de funciones, que son las más usadas y que inician simultáneamente una transacción de Meta4Object Engine y de base de datos. Se trata de BeginTransaction ( ) y EndTransaction ( ). Estas funciones están definidas como elementos del sistema, así que se puede tener acceso a ellas desde cualquier estructura de nodo.

Métodos transaccionales

Los métodos transaccionales son aquéllos que inician una transacción de Meta4Object Engine antes de iniciar su ejecución. Si se produce algún fallo en el método, se detendrá su ejecución, de modo que el funcionamiento del método origen que le llamó no se vea afectado, ya que éste prosigue su ejecución y realiza la instrucción siguiente a la llamada al elemento transaccional.

En el caso de que se produjese algún fallo en el elemento transaccional, la función devolvería como resultado M4_ERROR, de igual modo que en las variables por referencia, si las hay.

Un ejemplo de método transaccional es el método *Persist_tree ( )*.

BeginTransaction ( )

Esta función inicia una transacción de Meta4Object Engine y de base de datos. Si se produce un error cuando el Meta4Object Engine está ejecutando el código, la función evita que se detenga la ejecución.

Por ejemplo, la función ***Persist ( )*** puede fallar porque exista un cambio en el valor de la clave primaria.

Esta función se utiliza tanto cuando se lleva a cabo una transacción de base de datos como de ejecución.

EndTransaction (Commit / Rollback / Ejecutar postvalidación)

Esta función finaliza una transacción de Meta4Object y de base de datos. Hay que tener en cuenta que si se trabaja con base de datos, los datos no se graban físicamente en ella hasta que no se realice una transacción de base de datos en la que se guarden.

La función recibe un único argumento, con tres posibles valores:

- Commit. Graba permanentemente en la base de datos los cambios realizados.
- Rollback. En caso de producirse un fallo en la ejecución de una función, este argumento facilita que se deshagan todos los cambios realizados desde el inicio de la transacción; no graba en ningún momento.
- Ejecutar postvalidación. Este argumento comprueba el funcionamiento de las postvalidaciones. Tiene especial importancia en operaciones de importación de datos.

EJEMPLO DE SINTAXIS

```
EndTransaction (
   VALUE NUMBER Commit/RollBack
)
```

ExecuteSQL ( )

Esta función ejecuta una sentencia de APIs de SQL desde un Meta4Object. Acepta como argumentos cero u otros valores, cada uno de los cuales corresponde a un argumento.

ExecuteDirectSQL (ID_Conexión, Máscara,...)

Esta función ejecuta una sentencia SQL con una conexión lógica entre identificadores y una máscara de validación.

EJEMPLO DE SINTAXIS

```
ExecuteDirectSQL (
   VALUE NUMBER IdConnection, VALUE VARIABLE STRING Mask
)
```

ExecuteRealSQL (ID_Conexión,...)

Ejecuta una sentencia SQL con un identificador lógico de conexión.

EJEMPLO DE SINTAXIS

```
ExecuteRealSQL (
   VALUE NUMBER IdConnection
)
```

BeginDBTransaction

Esta función inicia una transacción de base de datos.

EndDBTransaction (Commit / Rollback / Ejecutar postvalidación)

Esta función finaliza una transacción de base de datos. Hasta que no se realiza una transacción de base de datos, la información no se graba físicamente.

La función recibe un único argumento, con tres posibles valores:

- Commit. Graba en la base de datos los cambios realizados.
- Rollback. Facilita, en caso de producirse un fallo en la ejecución de una función, que todos los cambios realizados desde el inicio de la transacción se deshagan.
- Ejecutar postvalidación. Este argumento comprueba el funcionamiento de las postvalidaciones. Esta posibilidad tiene su máximo significado en operaciones de importación de datos.

EJEMPLO DE SINTAXIS

```
EndDBTransaction (
   VALUE NUMBER Commit/RollBack
)
```

Load_blk ( )

Esta función realiza una carga de datos desde la base de datos sobre el bloque que reúne el conjunto de registros activo.

EJEMPLO DE SINTAXIS

```
Begin ( )
   Load_blk ( )
End ( )
```

Load_prg ( )

Esta función llama a la función Load_blk ( ) y realiza la carga de datos correspondiente a un nodo y a cada uno de los elementos conectados.

EJEMPLO DE SINTAXIS

```
Begin ( )
   Load_prg ( )
End ( )
```

Delete_Blk ( )

Esta función elimina de la base de datos todos los registros marcados como "borrados". Sólo elimina los registros borrados en el bloque activo del nodo desde el que se llama a la función. Para ejecutarla correctamente, siga estos pasos:

- Inicie una transacción en la base de datos física.
- Ejecute la función Detele_Blk ( ).
- Finalice la transacción grabando los cambios, para que se reflejen físicamente en la base de datos.

EJEMPLO DE SINTAXIS

```
Begin Transaction
   Delete_Blk ( )
End Transaction (Commit)
```

Delete_Prg ( )

Esta función elimina los registros que se hayan borrado en el bloque activo del nodo desde el que se llama a la función y todos aquellos que estén conectados y dependan de ellos.

Para ejecutar correctamente esta función, siga estos pasos:

- Inicie una transacción en la base de datos física.
- Ejecute la función DetelePrg ( )

- Para finalizar la transacción, grabe permanentemente los cambios para que se reflejen físicamente en la base de datos.

EJEMPLO DE SINTAXIS

```
Begin Transaction
   DeletePrg ( )
End Transaction (Commit)
```

Update_blk ( )

Esta función actualiza en la base de datos todos aquellos registros que hayan tenido cambios.

Sólo actualiza los registros que se hayan modificado en el bloque activo del nodo desde el que se llama a la función.

Para ejecutar correctamente esta función, siga estos pasos:

- Inicie una transacción en la base de datos física.
- Ejecute la función Update_blk ( )
- Para finalizar la transacción, grabe permanentemente los cambios en la base de datos.

EJEMPLO DE SINTAXIS

```
Begin Transaction
   UpdateBlk ( )
End Transaction (Commit)
```

Update_prg ( )

Esta función actualiza en la base de datos todos los registros que hayan tenido cambios, así como los registros relacionados.

Sólo actualiza aquellos registros modificados en el bloque activo del nodo desde el que se llama a la función y todos aquellos que dependan de ellos.

Para ejecutar correctamente la función, siga estos pasos:

- Inicie una transacción en la base de datos física.
- Ejecute la función Update_prg ( )

- Para finalizar la transacción, grabe permanentemente los cambios para que se reflejen en la base de datos.

EJEMPLO DE SINTAXIS

```
Begin Transaction
   UpdatePrg ( )
End Transaction (Commit)
```

Insert_blk ( )

Esta función inserta en la base de datos aquellos registros que hayan sido creados en el bloque activo del nodo desde el que se llama a la función.

Para ejecutar correctamente la función, siga estos pasos:

- Inicie una transacción en la base de datos física.
- Ejecute la función Insert_blk ( ).
- Finalmente, grabe permanentemente los cambios para que se reflejen en la base de datos.

EJEMPLO DE SINTAXIS

```
Begin Transaction
   InsertBlk ( )
End Transaction (Commit)
```

Insert_prg ( )

Esta función inserta en la base de datos aquellos registros que se hayan creado en el bloque activo del nodo desde el que se llama a la función y los que dependan de esos registros.

Para ejecutar correctamente esta función, siga estos pasos:

- Inicie una transacción en la base de datos física.
- Ejecute la función InsertPrg ( ).
- Finalmente, grabe permanentemente los cambios para que se reflejen físicamente en la base de datos.

EJEMPLO DE SINTAXIS

```
Begin Transaction
   InsertPrg ( )
End Transaction (Commit)
```

Persist_tree ( )

Esta función graba en la base de datos todos los cambios que se hayan efectuado en el bloque activo del nodo sobre el que esté posicionado, así como en los bloques de los nodos hijos con los que esté conectado.

La función ejecuta las siguientes acciones:

- Inicia una transacción en la base de datos física, para lo que realiza una llamada a la función Begin Transaction ( ).
- Posteriormente, realiza una de las siguientes acciones en función de la operación que se haya realizado previamente.
  - Borra de la base de datos física todos aquellos registros que se hayan marcado para borrar, bien desde la interfaz o mediante llamadas a la función Delete_Register ( ).
  - Actualiza en la base de datos física todos aquellos registros que hayan sido actualizados, bien desde la interfaz del programa o mediante llamadas a la función UpdateRegister ( ).
  - Inserta en la base de datos física todos aquellos registros que se hayan añadido desde la memoria, bien desde la interfaz del programa o mediante llamadas a la función InsertRegister ( ).
- Por último, cierra la transacción y, dependiendo de los resultados, ejecuta una de las siguientes acciones:
  - Graba permanentemente todos los cambios y los refleja en la base de datos, si éstos se han llevado a cabo satisfactoriamente.
  - Restaura el estado anterior, si no se ha podido realizar correctamente alguno de los pasos anteriores.

SetAutoloadMode (Modo)

La carga automática es una característica específica de los Meta4Objects, que permite que su contenido se cargue en memoria a medida que éste se va necesitando. De este modo, cuando se intenta tener acceso a un registro, el sistema comprueba si se ha visitado anteriormente:

- En caso negativo, procederá a la carga inicial de la información que contiene el registro.
- En caso contrario, el tiempo necesario para el acceso será más breve, ya que dispone de la información cargada en memoria.

Esta función aporta entre otras las siguientes ventajas:

- En el caso de un Meta4Object muy grande, no se necesita cargar completo en memoria, sino únicamente las partes que se visiten.
- El acceso a las partes ya visitadas con anterioridad es más rápido.

Los modos de carga automática se pueden establecer en dos niveles:

- ***Nivel de nodo:*** se explica en la función SetNodeAutoloadMode (Modo).
- ***Nivel de Meta4Object***: se puede indicar como nivel de carga el bloque. Desde el nivel de Meta4Object se puede especificar que se considere únicamente la configuración del nodo, de lo contrario, ésta no se tiene en cuenta.

Existen los siguientes tipos de carga automática:

- Se puede asignar a la carga automática el valor ***off***, lo que significa que la función está desactivada por defecto y en caso de que se quiera contar con ella, el usuario debe hacer la llamada.

  Si no se llama al Autoload (porque el Meta4Object no lo tenga definido previamente), el Meta4Object queda vacío, sin datos.
- Se puede establecer la carga automática en modo ***propagante***: cuando se accede a un bloque, sólo se carga la rama a la que pertenece dicho bloque, puesto que se necesita acceder a los hijos del bloque, pero no es necesario que estén cargados los demás bloques. Esto aporta una gran ventaja, ya que permite minimizar el número de transacciones a realizar.

  Este proceso se consigue llamando a la función Load_prg y no a la función Load_blk, que es la que se utilizaría cargar el bloque entero.
- Se puede establecer la carga automática a nivel de ***bloque***, lo que permite cargar en memoria un bloque entero.
- Se puede asignar a la carga automática ***nodesays***, lo que permite realizar la operación que el nodo tenga definida previamente.

EJEMPLO DE SINTAXIS

```
SetAutoloadMode (
   VALUE NUMBER Mode
)
```

SetNodeAutoloadMode (Modo)

Esta función establece la carga automática a nivel de nodo. La carga automática lleva a cabo las acciones que el nodo le indica.

Cada vez que la carga automática necesita realizar alguna acción, pregunta al nodo el tipo de carga que tiene asignado.

En los nodos puede haber diferentes clases de carga automática:

- Bloque: Esta clase de carga automática permite cargar un bloque entero.
- Propagante: Esta clase de carga automática permite cargar ciertos elementos dependientes de un bloque concreto, así como todos los elementos que dependen de ellos.
- Off: Esta clase de carga automática permite evitar la carga de un determinado nodo, independientemente de su motivo.

En cada nodo se puede tener un modo distinto de carga automática. Para obtener más información sobre este apartado, consulte las macros *Macros de autocarga*.

EJEMPLO DE SINTAXIS

```
SetNodeAutoloadMode (
   VALUE NUMBER Mode
)
```

GetAutoloadMode ( )

Esta función devuelve el modo de carga automática que tiene asignado el Meta4Object activo.

Para obtener más información sobre este apartado, consulte las macros Macros de autocarga.

GetNodeAutoloadMode (Modo)

Esta función devuelve el modo de autocarga que tiene asignado el nodo activo. Por defecto, devuelve el nodo activo cargado.

Para obtener más información sobre este apartado, consulte las macros definidas en la parte *Macros de autocarga*.

Funciones e-mind migradas a PeopleNet

DxYear (Fecha)

Esta función tiene como único argumento una fecha, de la cual devuelve el año a la que pertenece. Si el valor de la fecha es nulo, devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxYear (
   VALUE DATE AND TIME Date
)
```

DxCompare (Cadena1, Cadena2)

Compara dos cadenas que se indican como argumentos, que pueden tomar como valores fechas o números, e indica qué cadena es mayor.

La función puede devolver los siguientes valores:

- 1: si la primera cadena precede a la segunda.
- -1: si la primera cadena sigue a la segunda.
- 0: si las dos cadenas son iguales.

Si alguna de las cadenas tiene un valor nulo, la función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxCompare (
   VALUE VARIABLE STRING String1,
   VALUE VARIABLE STRING String2
)
```

DxDay (Fecha)

Devuelve el día de una fecha que se indica como argumento. Si la fecha es nula, esta función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxDay (
   VALUE DATE AND TIME Date
)
```

DxDaysOfMonth (Mes, Año)

Esta función devuelve el número de días que tiene un mes de un año concreto. Si alguno de los argumentos es nulo, la función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxDaysOfMonth (
   VALUE NUMBER Month,
   VALUE NUMBER Year
)
```

DxDate (Día, Mes, Año)

Esta función devuelve un valor con formato fecha, formado por día, mes y año. Si alguno de los argumentos es nulo, esta función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxDate (
   VALUE NUMBER Day,
   VALUE NUMBER Month,
   VALUE NUMBER Year
)
```

DxDate30 (Día, Mes, Año)

Esta función devuelve un valor formado por día, mes y año. A diferencia de la función *DxDate (Día, Mes, Año)*, ésta considera que todos los meses del año tienen 30 días. Si alguno de los argumentos es nulo, esta función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxDate30 (
   VALUE NUMBER Day,
   VALUE NUMBER Month,
   VALUE NUMBER Year
)
```

DxDateIso (Fecha)

Esta función devuelve como resultado un valor con formato fecha, formado por día, mes y año. A diferencia de la función *DxDate*, esta función devuelve el mismo valor utilizando el formato ISO yyyy-mm-dd. Si alguno de los argumentos es nulo, la función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxDateIso (
   VALUE DATE AND TIME Date
)
```

DxDateIsoD (Fecha)

Esta función devuelve un valor con formato fecha, formado por día, mes y año.

A diferencia de la función *DxDateIso*, ésta devuelve el mismo valor pero antecediendo a esta cadena el carácter "d", que indica la particularidad de esta función.

El resultado final se mostrará utilizando el formato { d yyyy-mm-dd }.

Si alguno de los argumentos es nulo, la función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxDateIsoD (
   VALUE DATE AND TIME Date
)
```

DxDateIsoTS (Fecha)

Esta función devuelve un valor con formato fecha, formado por día, mes y año.

A diferencia de la función *DxDateIso*, ésta devuelve el mismo valor, pero antecediendo a esta cadena el caracter "ts" que indica la particularidad de esta función.

El resultado final se mostrará utilizando el formato { ts yyyy-mm-dd }.

Si alguno de los argumentos es nulo, esta función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxDateIso TS (
   VALUE DATE AND TIME Date
)
```

DxInc (Valor)

Esta función permite incrementar un número o una fecha, indicado como argumento, en una unidad. Si el argumento es nulo, esta función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxInc (
   VALUE VARIANT Value
)
```

DxMonth (Fecha)

Esta función devuelve el mes correspondiente a una fecha indicada como argumento. Si dicha fecha es nula, la función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxMonth (
   VALUE DATE AND TIME Date
)
```

DxRoundCent (Valor)

Esta función redondea a centésimas el argumento indicado y muestra el número final con dos decimales. Si el argumento es nulo, esta función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxRoundCent (
   VALUE VARIANT Value
)
```

DxRound (Valor, Precisión)

Esta función redondear el primer argumento según la precisión indicada en el segundo. Realiza la misma operación que la función Round (Valor, Precisión).

Si el primer argumento es nulo, esta función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxRound (
   VALUE VARIANT Value
)
```

DxRound50 (Valor)

Esta función redondea el argumento indicado, en función del valor de éste. La posible casuística es la siguiente:

- Si la parte decimal del argumento está entre .01 y .49, redondeará la parte decimal a .50.
- Si la parte decimal del argumento está entre .51 y .99, redonderá la parte decimal a .0 e incrementará en uno la parte entera.

Si el argumento es nulo, esta función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxRound50 (
   VALUE VARIANT Value
)
```

DxTruncaCent (Valor)

Esta función trunca el número final para mostrarlo únicamente con dos decimales (centésimas).

Si el argumento es nulo, esta función devuelve como resultado un valor nulo.

EJEMPLO DE SINTAXIS

```
DxTruncaCent (
   VALUE VARIANT Value
)
```

Funciones para Meta4Objects

Todas las funciones correspondientes a este apartado tienen como ámbito de aplicación un Meta4Object, por lo que no se aplican a nivel de nodo. La sintaxis para ejecutar cualquiera de estas funciones es la siguiente:

```
nombre de Meta4Object o nombre de instancia ! Función ( )
```

Load ( )

La función carga desde la base de datos contenidos relativos a un Meta4Object, lo que significa que llama a la función Load_Prg dentro de todos los nodos raíz.

Persist ( )

Esta función graba físicamente en la base de datos contenidos relativos a un Meta4Object.

Release ( )

Esta función cierra un Meta4Object que esté en memoria. Tiene una limitación, ya que no se puede ejecutar sobre el mismo Meta4Object desde el que se ejecuta, sino que se tiene que ejecutar sobre otro.

Unload ( )

Descarga todos los datos que se tengan en memoria para poder ejecutar con éxito la función *Load ( )* y poder cargar nueva información.

Esta función tiene una limitación, ya que no se puede ejecutar sobre el mismo Meta4Object desde el que se ejecuta, sino que debe realizarse sobre cualquier otro.

IsDataCacheable ( )

Permite conocer si un determinado Meta4Object tiene configurados los datos como almacenables en la memoria caché. La propiedad *Data Cacheable* puede tener los siguientes valores:

- 1: indica que los datos se pueden almacenar en la caché.
- 0: indica que los datos no se pueden almacenar en la caché.

AddAsPartner ( )

Cuando se realizan transacciones cliente/servidor, en ocasiones es necesario utilizar Meta4Objects compañeros, puesto que es posible que un Meta4Object requiera el uso de otros, no sólo cuando se realizan llamadas de nivel 2.

Si posteriormente se quiere eliminar de la lista cualquiera de estos Meta4Objects seleccionados como compañeros, se debe utilizar la función *RemoveAsPartner ( )*.

RemoveAsPartner ( )

Para realizar transacciones cliente/servidor, se necesitan Meta4Objects compañeros. Esta función elimina de la lista determinados Meta4Objects compañeros que hayan sido añadidos previamente mediante la función *AddAsPartner ( )*.

SetProperty (Propiedad, Valor)

Los Meta4Objects pueden tener propiedades. Esta función asigna valores a las propiedades existentes, o crea otras nuevas, como por ejemplo, la fecha de ejecución.

GetProperty (Propiedad)

Con esta función se pueden conocer los valores que tienen asignadas las propiedades existentes, como por ejemplo, la fecha de ejecución.

EJEMPLO DE SINTAXIS

```
Meta4Object!GetProperty ("IdPropiedad")
```

LoadFromFile (NombreArchivo)

Esta función carga datos de un Meta4Object desde un archivo. No se puede aplicar sobre el Meta4Object que se está ejecutando, ya que devolvería un código de error.

PersistToFile ([NombreArchivo])

Esta función guarda datos de un Meta4Object en un archivo. Si el archivo especificado no existe, la función lo crea.

Si la función se ejecuta correctamente, devuelve el nombre; si se produce algún error, devuelve nulo. El nombre de archivo es opcional, ya que si no se indica, se asigna uno por defecto.

CheckModify ( )

Esta función indica si ha habido alguna modificación en el Meta4Object. Se utiliza con frecuencia en herramientas que deben preguntar al usuario si ha habido cambios en un Meta4Object antes de proceder a cerrarlo, para que, en caso afirmativo, se graben físicamente en la base de datos.

GetHandle ( )

Esta función retorna el identificador numérico único de una instancia de Meta4Object. Este identificador numérico se calcula en el momento que se crea la instancia del Meta4Object y no cambia a lo largo de toda la vida de esa instancia. Este número sirve para identificar de manera unívoca una instancia de Meta4Object. En ln4 puede utilizarse con otras funciones de ln4 como InsertChannel.

Esta función no tiene argumentos y se aplica al Meta4Object sobre el que se esté trabajando.

Biblioteca básica de funciones de nivel 2

OpenChannel (Meta4Object, IDInstancia, Ámbito, ModoApertura)

Esta función abre una instancia de Meta4Object utilizando una llamada de Meta4Object Engine de nivel 2.

Si el Meta4Object no está cargado, la función permite sobrescribir los

argumentos *Ambito* y *ModoApertura*, es decir, el ámbito y el modo en el que se abre el Meta4Object, cuyos posibles valores están definidos como constantes.

Los argumentos correspondientes a esta función son los siguientes:

- Meta4Object

  Contiene el identificador del Meta4Object del que se va a abrir una nueva instancia.

- IDInstancia

  Contiene el identificador de la instancia.

- Ámbito

  Ámbito de la instancia abierta. Indica la política de compartición. Los posibles valores que puede tomar este argumento están definidos como constantes.

  Para más información consultar el apartado *Macros LN4*.

- ModoApertura

  Indica el modo en que se abre una instancia en el Meta4Object. Los posibles valores que puede tomar este argumento están definidos como constantes.

  Para más información, consulte el apartado *Macros LN4*.

EJEMPLO DE SINTAXIS

```
OpenChannel (
   VALUE VARIABLE STRING Meta4Object,
   VALUE VARIABLE STRING IdInstance,
   VALUE NUMBER Scope,
   VALUE NUMBER OpenMode
)
```

OpenChannelEx (Meta4Object, IdInstancia, Ambito, ModoApertura, [OrganizaciónID], [TipoOrganización], [CSTipoAcceso])

Esta función modifica el tipo de organización, tipo de acceso, ámbito, modo, etc., con el que se abre un Meta4Object que esté cargado en memoria.

La descripción que está en la base de datos se utiliza para cargar el Meta4Object, que tendrá como características aquellas que estén definidas en la base de datos.

Los argumentos correspondientes a esta función son los siguientes:

- Meta4Object

  Contiene el identificador del Meta4Object del que se va a abrir una nueva instancia.

- IdInstancia

  Contiene el identificador de la instancia.

- Ambito

  Ámbito de la instancia abierta. Indica la política de compartición. Los posibles valores que puede tomar este argumento están definidos como constantes.

  Para más información, consulte el apartado *Macros LN4*.

- ModoApertura

  Indica el modo en que se abre una instancia en el Meta4Object. Los posibles valores que puede tomar este argumento están definidos como constantes.

  Para más información consultar el apartado *Macros LN4*.

- IDOrganización

  Es un argumento opcional e indica el identificador de la organización del Meta4Object.

- TipoOrganización

  Es un argumento opcional e indica el tipo de organización del Meta4Object.

- CSTipoAcceso

  Es un argumento opcional e indica el tipo de acceso del Meta4Object.

InsertChannel

Esta función crea un nuevo enlace de nivel 2 en el Meta4Object sobre el que se esté trabajando. El enlace de nivel 2 se hace a la instancia dada por el primer argumento que es un identificador numérico único que se debe haber obtenido con una llamada a la función GetHandle. Este enlace de nivel 2 se asocia al identificador de instancia dado por el segundo argumento. Si ya existe un enlace de nivel 2 para ese identificador de instancia, no se realiza la operación y se devuelve un error.

Los argumentos correspondientes a esta función son los siguientes:

- **ai_iM4ObjHandle**: número que representa un identificador numérico único de una instancia de Meta4Object.
- **ai_sInstanceId**: cadena que representa el identificador de la instancia de nivel 2.

CloseChannel (IdInstancia)

Esta función cierra el Meta4Object correspondiente a una instancia, cuyo identificador se ha indicado como argumento.

EJEMPLO DE SINTAXIS

```
CloseChannel (
   VALUE VARIABLE STRING IdInstance
)
```

DefineInstance (Instancia, Meta4Object, PolíticaCarga, PolíticaCompartición)

Esta función define una instancia de un Meta4Object de forma dinámica. Sólo funciona para ejecuciones Just In Time (JIT), y sus argumentos son los siguientes:

- Instancia

  Corresponde al identificador de la instancia.

- Meta4Object

  Contiene el identificador del Meta4Object del que se va a abrir una instancia.

- PolíticaCarga

  Indica la política de carga del Meta4Object.

- PolíticaCompartición

  Muestra la política de compartición de la instancia.

EJEMPLO DE SINTAXIS

```
DefineInstance (
   VALUE VARIABLE STRING Instance,
   VALUE VARIABLE STRING Meta4Object,
   VALUE NUMBER PolíticaCarga,
   VALUE NUMBER SharePolicy
)
```

DefineInstanceEx (Instancia, Meta4ObjectNombre, PolíticaCarga, PolíticaCompartición, [IDOrganización], [TipoOrganización], [CSTipoAcceso])

Esta función define una instancia correspondiente a un Meta4Object de forma dinámica. Además, permite definir la organización, tipo de organización y tipo de acceso.

Sólo funciona para ejecución Just In Time (JIT).

Los argumentos correspondientes a esta función son los siguientes:

- Instancia

  Corresponde al identificador de la instancia.
- Meta4Object

  Contiene el identificador del Meta4Object del que se va a abrir una instancia.
- PolíticaCarga

  Indica la política de carga del Meta4Object.
- PolíticaCompartición

  Contiene la política de compartición.
- IDOrganización

  Contiene el identificador unívoco de la organización definida.
- TipoOrganización

  Indica el tipo de organización.
- CSTipoAcceso

  Indica el tipo de acceso al Meta4Object.

EraseAllL2Instances

Esta función permite eliminar todas las instancias que se hayan abierto con llamadas de nivel 2.

Funciones de log

Estas funciones permiten escribir en el sistema log mensajes de diverso tipo (error, aviso, depuración, etc.).

SetLog (Tipo, Módulo, Submódulo, Código, [Argumento])

Esta función añade un mensaje de tipo error o aviso al sistema log.

Los argumentos correspondientes a esta función son los siguientes:

- Tipo

  Contiene el tipo de mensaje. Puede ser de error, aviso o depuración.

- Módulo

  Identifica el módulo de error o aviso. El valor de este argumento puede variar entre 0 y 255.

- Submódulo

  Identifica el submódulo de error o aviso. El valor de este argumento puede variar entre 0 y 255.

- Código

  Identifica el código de error o aviso. El valor de este argumento puede variar entre 0 y 65.535.

- Argumento

  Es un argumento opcional y corresponde a los argumentos del mensaje de error definido en el Meta4Object de errores. Es una descripción textual según el tipo de error o aviso.

EJEMPLO DE SINTAXIS

```
SetLog (
   VALUE NUMBER Type,
   VALUE NUMBER Module,
   VALUE NUMBER SubModule,
   VALUE VARIABLE STRING Code,
   [VALUE VARIANT Argument]
)
```

GetLog (Posición, &Tipo, &Módulo, &Submódulo, &Código, &TextoAdicional)

Esta función lee un mensaje de error del sistema de errores.

Los argumentos correspondientes a esta función son los siguientes:

- Posición

  Contiene la posición del mensaje. Se pueden dar dos situaciones:

  - Si la posición es positiva, adquiere valores desde el principio, es decir, los más recientes.
  - Si la posición es negativa, adquiere valores desde el final, es decir, los más antiguos.

- &Tipo

  Contiene por referencia el tipo de mensaje (de error o de aviso).

- &Módulo

  Identifica por referencia el módulo de error o aviso. El valor de este argumento puede variar entre 0 y 255.

- &Submódulo

  Identifica por referencia el submódulo de error o aviso. El valor de este argumento puede variar entre 0 y 255.

- &Código

  Identifica por referencia el código de error o aviso. El valor de este argumento puede variar entre 0 y 65.536.

- &TextoAdicional

  Contiene por referencia el mensaje de error definido en el Meta4Object de errores, según el tipo de error o aviso.

EJEMPLO DE SINTAXIS

```
GetLog (
   VALUE NUMBER Position,
   REFERENCE NUMBER Type,
   REFERENCE NUMBER Module,
   REFERENCE NUMBER SubModule,
   REFERENCE NUMBER Code,
   REFERENCE VARIABLE STRING AditionalText
)
```

PurgeLog (NumElimina)

Esta función elimina mensajes de error del sistema de errores, ya que están en memoria.

Se pueden dar dos situaciones:

- Si el valor es positivo, la función elimina empezando desde el principio, es decir, los errores más recientes.
- Si el valor es negativo, la función elimina empezando desde el final, es decir, los errores más antiguos.

EJEMPLO

```
PurgeLog(
   VALUE NUMBER NumPurges
)
```

GetLogSize ( )

Esta función devuelve el número de mensajes de error que están en la cola de errores en memoria.

GetErrorString (Módulo, Submódulo, Código, [Argumento 1])

Esta función devuelve el texto asociado a un error, formateando la cadena indicada en función de los argumentos introducidos y del idioma en que esté.

Los argumentos correspondientes a la función son:

- Módulo

  Identifica el módulo de error o aviso. El valor de este argumento puede variar entre 0 y 255.
- Submódulo

  Identifica el submódulo de error o aviso. El valor de este argumento puede variar entre 0 y 255.
- Código

  Identifica el código de error o aviso. El valor de este argumento puede variar entre 0 y 65.536.
- [Argumento 1]

Indica el primer argumento que se debe sustituir.

EJEMPLO

```
GetErrorString(
   VALUE NUMBER Module,
   VALUE NUMBER SubModule,
   VALUE NUMBER Code,
   [VALUE VARIANT Argument1]
)
```

InsertChannel(ai_iM4ObjHandle, ai_sInstanceId)

Esta función crea un nuevo enlace de nivel 2 en el Meta4Object sobre el que se esté trabajando. El enlace de nivel 2 se hace a la instancia dada por el primer argumento que es un identificador numérico único que se debe haber obtenido con una llamada a la función GetHandle. Este enlace de nivel 2 se asocia al identificador de instancia dado por el segundo argumento. Si ya existe un enlace de nivel 2 para ese identificador de instancia no se realiza la operación y se devuelve el mensaje de error M4_ERROR, en caso contrario, si la operación ha sido correcta retorna el M4_SUCCESS

Los argumentos correspondientes a la función son:

- **ai_iM4ObjHandle:** Número que representa un identificador numérico único de una instancia de Meta4Object.
- **ai_sInstanceId**: Cadena que representa el identificador de la instancia de nivel 2.

Funciones herramienta

Pivot (IDMeta4Object, IDHost, DominioDestino, [Índice])

Esta función pivota un Meta4Object a través de un dominio de metadatos. La acción de pivotado consiste en cambiar las filas por columnas.

Los metadatos correspondientes a cada uno de los campos tienen una propiedad llamada campo de dominio, que debe contener el nombre de un elemento. Estos elementos del campo de dominio suelen estar marcados como tipo interno componentes del dominio, para distinguirlos del resto de los elementos. Si coincide el valor, se actúa sobre él y se indica el valor.

Los argumentos correspondientes a esta función son:

- IDMeta4Object

  En él se especifica el Meta4Object donde se encuentra el nodo con los registros a pivotar. Se pueden dar dos situaciones:

  – Si el argumento está vacío, "", se asume que el nodo origen está en el propio Meta4Object que se encuentra activo.

  – Si el argumento no está vacío, el nodo donde se encuentra el elemento pivote debe tener definido el uso del otro Meta4Object por la interfaz de nivel 2, de modo que el argumento IDMeta4Object pueda corresponderse con un alias del Meta4Object del nodo donde está el elemento pivote, o de igual modo se pueda corresponder con una instancia del Meta4Object donde está el elemento pivote.

- IDHost

  Es un argumento que nunca puede estar vacío. En él se especifica el nodo donde se encuentran los registros que se van a pivotar. Se pueden dar dos situaciones:

  – Si existe IDMeta4Object como primer argumento, especifica el nombre del nodo de ese Meta4Object donde están los registros que se van a pivotar.

  – Si no existe IDMeta4Object como primer argumento, el argumento especifica uno de los siguientes elementos:

    – El alias del nodo donde están los registros a pivotar para el nodo donde se ejecuta la función de pivotado

    – El nombre de la estructura de nodo donde están los registros que se van a pivotar

    – El nombre del nodo, si no es ninguno de los anteriores

- DominioDestino

  Especifica el DM destino sobre el que se realiza el pivotado. Está compuesto por *DMD_fields*. Si un elemento del DM destino no pertenece a este DM, no se lleva a cabo la aplicación del pivotado para él.

  Si el valor de este argumento es (""), se realiza el pivotado para los elementos de cualquier DM.

- Índice

  Es un argumento opcional. Especifica el índice que se utilizará para ordenar los registros que se van a pivotar.

  Además, indica cuál es la clave primaria que se va a utilizar para los registros. Si este argumento no aparece, se emplea el índice por defecto del nodo con los registros que se van a pivotar.

Funcionamiento del pivotado

En el caso de un nodo que contiene un conjunto de registros expuestos en forma de acumulado corto, el algoritmo de pivotado genera en un nodo destino el mismo número de registros que existen en el nodo origen con diferente clave primaria.

La clave primaria del nodo origen viene determinada por un índice que debe estar definido en ese nodo. Los elementos del índice son los elementos que marcan la clave primaria.

Lo primero que hace el algoritmo es ordenar el nodo origen según un índice concreto. Esto no es necesario, pero hace que el algoritmo sea más rápido posteriormente.

Después de esto, ya es posible pivotar el registro. Para saber cuál es la columna del nodo origen que indica el componente DM, se debe marcar únicamente ese elemento con el tipo interno *DMD_component*.

El algoritmo recorre todas las columnas del nodo origen, excepto la del *DMD_component*, intentando pivotar su contenido. Para ello:

1. El DM de la columna destino debe ser igual al indicado como tercer argumento, o a cualquier otro si ese argumento está vacío.
2. El *DMD_component* de la columna destino debe ser igual al valor de la columna *DMD_component* origen (marcada como 18).
3. El *DMD_field* de la columna destino debe coincidir con el nombre de la columna origen que se va a pivotar.

El algoritmo recorre todas las columnas origen intentando pivotarlas sobre cualquier columna destino que cumpla las condiciones especificadas anteriormente.

No existe límite en las columnas destino que pueden cumplir la condición de búsqueda.

Un pivotado incremental es la operación de asignar valor sólo en aquellas celdas destino cuyo valor sea distinto al que se desea asignar, no así en aquellas celdas en las que coincida.

Si una celda destino ya existe y tiene un valor idéntico al que se va a asignar, no se llevará a cabo la asignación, lo que permitirá realizar pivotados incrementales.

EJEMPLO

Considere la siguiente tabla origen.

ID_Concept	ID_HR	OR_HR_Role	Valor
1001	4497	1	5
1011	4497	1	8
1021	4497	1	4
1001	4498	1	6
1011	4498	1	16
1001	2234	1	2
1021	2234	1	32

Las columnas marcadas como clave primaria mediante el índice serían ID_HR y OR_HR_ROLE. La columa ID_Concept tiene 18 como tipo interno.

Considere una tabla destino del siguiente tipo.

ID_HR	OR_HR_Role	CELL1	CELL2	CELL3	CELL4
2234	1	2	<NULL>	32	2
4497	1	5	8	4	5
4498	1	6	16	<NULL>	6

Esta tabla sería el resultado de aplicar la función de pivotado sobre un DM (por ejemplo *payroll*), asumiendo lo siguiente:

- El elemento *Cell1* tiene un DM *payroll*, un *DMD_component* 1001 y un *DMD_field value*.
- El elemento *Cell2* tiene un DM *payroll*, un *DMD_component* 1011 y un *DMD_field value*.
- El elemento *Cell3* tiene un DM *payroll*, un *DMD_component* 1021 y un *DMD_field value*.
- El elemento *Cell4* tiene un DM *payroll*, un *DMD_component* 1001 y un *DMD_field value*.

Por ejemplo, si hubiese otra columna en el origen, se pivotaría sobre el elemento *Cell5*, si éste tuviese un DM *payroll*, un *DMD_component* 1011 y un *DMD_field moneda*.

EJEMPLO DE SINTAXIS

```
Pivot(
   VALUE VARIABLE STRING Meta4Object,
   VALUE VARIABLE STRING HostId,
   VALUE VARIABLE STRING TargetDMD,
   [VALUE NUMBER Index]
)
```

LookUp (PCP1, PCP2...ES1, SCP1, SI1, SCP2..., NúmeroSecundario, ElementoFiltroaAplicar, NodoUsuario, FechaInicioApl, FechaFinApl, FechaInicioCorr, FechaFinCorr)

Esta función está orientada a proporcionar un servicio funcional, consistente en filtrar un nodo. Se la puede llamar desde las funciones *LIST* y *Validate*, que son métodos generados con la herramienta Diseñador del modelo de datos, mediante los cuales se pueden filtrar nodos.

Sus argumentos correspondientes son:

- **PCPn**: es un argumento opcional, y contiene la clave primaria que permite conocer el contenido de los elementos primarios.
- **ESn**: es un argumento opcional, que permite conocer el contenido de los elementos secundarios. Siempre se indica en números pares, ya que debe contener el identificador del elemento y el contenido por el que se va a buscar. En este caso, el argumento SIn contiene el identificador del elemento.
- **SCPn**: este argumento opcional indica el contenido del elemento por el que se va a buscar. El identificador del elemento se indica en el argumento *SIn*.
- **NúmeroSecundario**: este argumento indica el número de parejas de elementos secundarios que existen. No es necesario indicar el número de elementos primarios, ya que se obtiene de los secundarios.
- **ElementoFiltroaAplicar**: este argumento es obligatorio y de tipo cadena; en caso de no ser así, generaría un error. Contiene el identificador del elemento que se quiere invocar, el cual permite generar la sentencia SQL que se pretende ejecutar sobre el Meta4Object.

  Este valor constituye realmente un método al cual se invoca. Su comportamiento varía dependiendo de si el Meta4Object se puede almacenar en la caché.

- **NodoUsuario**: este argumento es el identificador de un nodo; sirve para añadir una funcionalidad adicional a la función *ElementoFiltroaAplicar*, por ejemplo, incluir un filtro adicional.

  En este caso, el argumento indica el identificador del nodo donde va a estar el elemento que calcula ese filtro adicional.

  El identificador del elemento que se va a invocar debe ser *APPLY_FILTER*. Este argumento puede tener como valor nulo o " ", en cuyo caso no se tendría en cuenta.
- **FechaInicioApl**: este argumento filtra datos utilizando como criterio la fecha de inicio. Siempre aparece, aunque su valor pueda ser nulo, " " o cualquier otro dato que no sea fecha, en cuyo caso no se tendría en cuenta.
- **FechaFinApl**: este argumento filtra datos por fecha de fin y siempre aparece, aunque su valor puede ser nulo, " " o cualquier otro dato que no sea fecha, en cuyo caso no se tendría en cuenta.
- **FechaInicioCorr**: este argumento filtra datos por fecha de inicio de corrección y siempre aparece, aunque su valor puede ser nulo, " " o cualquier otro dato que no sea fecha, en cuyo caso no se tendría en cuenta.
- **FechaFinCorr**: este argumento filtra datos por fecha de fin de corrección y siempre aparece, aunque su valor puede ser nulo, " " o cualquier otro dato que no sea fecha, en cuyo caso no se tendría en cuenta.

Aplicación de fechas

Existen dos modos de aplicación:

- Si el Meta4Object no se puede almacenar en la memoria caché, se comprueba si existen elementos marcados con un tipo interno determinado de filtro de fecha. Si es así, se rellenan las fechas, siempre que existan, con los valores que aparecen en los argumentos *FechaInicioApl, FechaFinApl, FechaInicioCorr* y *FechaFinCorr*, para que posteriormente la base de datos lógica los aplique en las sentencias que ejecute.
- Si el Meta4Object se puede almacenar en la caché, se compone un filtro en RAM similar a los filtros que aplicaría la BDL.

Filtro en RAM

Independientemente de si el Meta4Object se puede almacenar en la caché o no, se construye una parte del filtro en RAM. Esta parte del filtro es para la parte *LIKE*, es decir, se toman los argumentos secundarios indicados y se aplica un filtro en RAM para cada uno de ellos.

Para cada pareja de secundarios que exista, se construye una parte del filtro en la que se concatenan mediante el operador booleano AND.

Tratamiento del argumento ElementoFiltroaAplicar

Posteriormente, se invoca al elemento contenido en *ElementoFiltroaAplicar*, que actuará de una forma u otra en función de si el Meta4Object se puede almacenar en la caché.

A *ElementoFiltroaAplicar* se le asignan los argumentos primarios y un indicador que determina si el elemento se puede almacenar en la caché:

- 0: indica que el elemento no se puede almacenar en la caché.
- 1: indica que el elemento se puede almacenar en la caché.

La función *ElementoFiltroaAplicar* aplica un filtro al nodo donde se invoca. Si el Meta4Object no se puede almacenar en la caché, genera un filtro SQL con el que se rellena el *SysSentence*. Si se puede almacenar en la caché, el método crea un filtro en RAM, parecido al SQL, no rellena el *SysSentence* y devuelve como resultado de su ejecución el filtro en RAM que se aplicará posteriormente.

SysSentence es un elemento especial que permite sobrescribir una sentencia determinada asociada a una estructura de nodo o añadir un nuevo filtro que se concatena a la sentencia principal de la estructura de nodo.

Tratamiento del elemento definido por el usuario

Como siguiente paso se invoca el elemento definido por el usuario. Si el nodo elegido por el usuario hace las mismas cosas que el *ElementoFiltroaAplicar*, se indicaría aquí. Se define en el argumento *NodoUsuario*.

Por último, se indicaría si el Meta4Object se puede almacenar en la caché o no. Es un método que no va a tener más que un argumento que puede ser del siguiente tipo:

- 0: indica que el Meta4Object no se puede almacenar en la caché.
- 1: indica que el Meta4Object se puede almacenar en la caché.

Con ese método, si el Meta4Object no se puede almacenar en la caché, el usuario tendrá que aplicar un filtro o sino devolver un filtro en RAM para que posteriormente se aplique.

Funcionalidad añadida de LookUp

La función *LookUp* concatena con el operador AND todas las operaciones anteriores. Si el Meta4Object se pueda almacenar en la caché, habrá varios AND y si no se puede almacenar en la caché, habrá uno que será el LIKE, que permitirá buscar una cadena concreta indicada como argumento.

Por último, se procede a realizar la carga del bloque donde se ha invocado la función *LookUp*.

Si el Meta4Object no se puede almacenar en la caché, se invoca al elemento

Load_prg, con lo cual se desencadena la carga del bloque de hijos filtrado por los filtros SQL que hayan podido aplicar los elementos *APPLY_FILTER*.

Si el Meta4Object se puede almacenar en la caché, la función comprueba si la información está cargada en memoria.

Si no lo está, se llama a la función *Load ( )* del Meta4Object. Después de estas operaciones,el Meta4Object queda cargado de un determinado modo.

Por último, se aplican dos filtros en RAM:

- Filtro en RAM del LIKE, para realizar búsquedas literales de un patrón indicado.
- Resto de filtros, sólo si el Meta4Object se almacena en la caché.

Como resultado final, la función *LookUp* devuelve el número de registros que quedan después de aplicar los filtros.

La función tiene dos excepciones:

- Si el método al que tenemos que invocar con *Apply_list* y el Meta4Object no se puede almacenar en la caché, no se invoca el método.
- La función borra los filtros de anteriores ejecuciones del *LookUp* antes de aplicar los filtros nuevos.

EJEMPLO DE SINTAXIS

```
LookUp (
   VALUE VARIABLE STRING PrimaryPK1,
   VALUE VARIABLE STRING SecondaryPK,
   VALUE VARIABLE STRING SecondaryValue,
   VALUE NUMBER SecondaryNumber,
   VALUE VARIABLE STRING ApplyFiterItem,
   VALUE VARIABLE STRING UserNode,
   VALUE DATE AND TIME AppStartDate,
   VALUE DATE AND TIME AppEndDate,
   VALUE DATE AND TIME CorrStartDate,
   VALUE DATE AND TIME CorrEndDate
)
```

EJEMPLO 1

En este ejemplo se utiliza el método *LIST*, que invoca a la función *LookUp*. Los argumentos que se indican son los siguientes:

- ARG_ID_OBJECT: Indica el valor de la clave primaria. El método *LIST* dispone además de otro argumento que no se muestra en este ejemplo y que se utiliza para indicar el valor de la clave primaria secundaria.
- ID_Índice: Contiene el identificador de la columna en la que se va a realizar la búsqueda.
- ARG_ID_INDEX: Indica el valor de la cadena buscada.
- APPLY_LIST: Indica el método que se invoca desde la función *LookUp*.
- GetArgument (1): Indica el nodo activo del usuario.
- GetArgument (2): Indica la fecha de inicio.
- GetArgument (3): Contiene la fecha de fin.

El código correspondiente de este ejemplo es:

```
'LIST
iReturn = LookUp(ARG_ID_OBJECT,  "ID_INDEX", ARG_ID_INDEX, 1,
"APPLY_LIST", GetArgument(1), GetArgument(2), GetArgument(3), "",
"")
RETURN iReturn
```

EJEMPLO 2

En este ejemplo, el método *APPLY_FILTER_LIST* invoca la función *LookUp*.

En la primera parte del código, hasta la sentencia *ELSE* se crea un filtro RAM, y a partir de esa sentencia, se ejecuta la sentencia *SELECT* de SQL. Ambas operaciones se realizan sobre el Meta4Object.

```
'APPLY_FILTER_LIST
iLastArg = GetArgumentNumber()
sSQLParam = ""
sArgParam = ""
sRAMParam = ""
sEmpty = ""

sFROM = "FROM &SDD_INDEX SDD_INDEX "

If GetArgument(iLastArg) = TRUE Then

      'Template
      'If IsNull(ARG) <> TRUE And Arg <> sEmpty Then
      '     sValue = "ITEM = " + Chr({HoldChr1}) +
             ToString(ARG, 0) + Chr({HoldChr2})
```

```
'     If sRAMParam <> sEmpty Then
'          sRAMParam = " AND " + sRAMParam
'     End If
'     sRAMParam = sRAMParam + sValue
'EndIf

If IsNull(ARG_ID_OBJECT) <> TRUE And ARG_ID_OBJECT <>
sEmpty Then
     sValue = "ID_OBJECT = " + CHR(34) +
      ToString(ARG_ID_OBJECT, 0) + CHR(34)
     If sRAMParam <> sEmpty Then
          sRAMParam = sRAMParam + " And "
     EndIf
     sRAMParam = sRAMParam + sValue
EndIf

sReturn = sRAMParam

Else

     'Template
      'If IsNull(ARG) <> TRUE And Arg <> sEmpty Then
      '      sValue = "ALIAS.FIELD = ?(type, prec, scale)"
      '      If sSQLParam <> sEmpty Then
      '          sSQLParam = " AND " + sSQLParam
      '     EndIf
      '     sSQLParam = sSQLParam + sValue
      '     If sArgParam <> sEmpty Then
      '          sArgParam = "|" + sArgParam
      '     EndIf
      '     sArgParam = sArgParam + ToString(ARG, 0)
      'EndIf

     If IsNull(ARG_ID_OBJECT) <> TRUE And ARG_ID_OBJECT <>
     sEmpty Then
          sValue = "SDD_INDEX.ID_OBJECT= ?(2, 30, 0)"
          If sSQLParam <> sEmpty Then
               sSQLParam = sSQLParam + " And "
          EndIf
          sSQLParam = sSQLParam + sValue
          If sArgParam <> sEmpty Then
```

```
                sArgParam = sArgParam + "|"
            EndIf
            sArgParam = sArgParam + ToString(ARG_ID_OBJECT, 0)
        EndIf

        If sSQLParam <> "" Then
            sSQLParam = " WHERE " + sSQLParam
        EndIf

        SYS_SENTENCE_A = sFROM + sSQLParam
        SYS_PARAM_A = sArgParam
        sReturn = ""
    EndIf

    RETURN sReturn
```

Funciones para transacciones de memoria

CheckPointRegister ( )

Esta función realiza transacciones en memoria e inicia otras nuevas a nivel de registro.

Cuando se están realizando cambios en el Meta4Object, se guarda el estado en el que estaba antes de llevar a cabo la última transacción. De este modo, si se ejecuta la función *CheckPointRegister*, los cambios realizados se graban en el Meta4Object.

Si se realiza un *UndoRegister*, se deshacen estos cambios. De este modo, el estado del Meta4Object es el mismo que después de ejecutar la función *CheckPointRegister* por última vez.

CheckPointBlock ( )

Esta función realiza transacciones en memoria e inicia otras nuevas a nivel de bloque.

Cuando se están realizando cambios en el Meta4Object se guarda el estado en el que estaba antes de llevar a cabo la última transacción. De este modo, si se ejecuta la función *CheckPointBlock*, los cambios realizados se graban en el Meta4Object.

Si se realiza un *UndoBlock* se deshacen estos cambios. De este modo, el estado del Meta4Object es el mismo que después de ejecutar la función *CheckPointBlock* por última vez.

CheckPointNode ( )

Esta función realiza transacciones en memoria e inicia otras nuevas a nivel de nodo.

Cuando se están realizando cambios en el Meta4Object se guarda el estado en el que estaba antes de llevar a cabo la última transacción. De este modo, si se ejecuta la función *CheckPointNode*, los cambios realizados se graban en el Meta4Object.

Si se realiza un *UndoNode*, se deshacen estos cambios. De este modo, el estado del Meta4Object es el mismo que después de ejecutar la función *CheckPointNode* por última vez.

CheckPointChannel ( )

Esta función realiza transacciones en memoria e inicia otras nuevas a nivel de Meta4Object.

Cuando se están realizando cambios en el Meta4Object, se guarda el estado en el que estaba antes de llevar a cabo la última transacción. De este modo, si se ejecuta la función *CheckPointChannel*, los cambios realizados se graban en el Meta4Object.

Si se realiza un *UndoChannel* se deshacen estos cambios. De este modo, el estado del Meta4Object es el mismo que después de ejecutar la función *CheckPointChannel* por última vez.

UndoRegister ( )

Esta función deshace los últimos cambios que se hayan realizado en el Meta4Object a nivel de registro.

UndoBlock ( )

Esta función deshace los últimos cambios que se hayan realizado en el Meta4Object a nivel de bloque.

UndoNode ( )

Esta función deshace los últimos cambios que se hayan realizado en el Meta4Object a nivel de nodo.

UndoChannel ( )

Esta función deshace los últimos cambios que se hayan realizado en el Meta4Object a nivel de Meta4Object.

Funciones de nivel 1 para metadatos

Las funciones de nivel 1 permiten obtener información de metadatos, y en todas ellas el argumento filtro se entiende como un criterio según el cual se establece una selección de elementos determinada.

ExistNodeItemIndex (Nodo, Pos, [Filtro]) o (Nodo, IDElemento)

Esta función devuelve el índice correspondiente a un elemento si existe. En esta función se entiende por índice el identificador unívoco que posee cada uno de los elementos.

Los argumentos correspondientes a esta función:

- **Nodo**: contiene el nombre del nodo que se va a utilizar.
- **IDElemento**: este argumento puede tomar uno de dos valores posibles:
  - Nombre del elemento al que se hace referencia.
  - Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro.
- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
ExistNodeItemIndex (
VALUE VARIABLE STRING Node,
VALUE VARIANT Item,
[VALUE NUMBER Filter]
)
```

No se genera ningún tipo de aviso si no existe el elemento.

GetChannelId

Esta función devuelve el identificador del Meta4Object activo.

GetThisNodeId

Esta función devuelve el identificador del nodo activo.

GetNodeNumberOfItems (Nodo, Filtro)

Esta función devuelve el número de elementos que existen en el nodo que se pasa como primer argumento.

EJEMPLO DE SINTAXIS

```
GetNodeNumberOfItems (
   VALUE VARIABLE STRING Node,
   VALUE NUMBER Filter
)
```

El filtro que hay que aplicar se usa para visualizar los elementos de un ámbito específico y de un tipo específico, o de una combinación de los dos.

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Filtro**: contiene el filtro que se va a aplicar sobre el nodo.

Para esto, use los macros de tipo y de ámbito disponibles. Los posibles valores para el filtro son los siguientes:

- M4_SCOPE_ALL: Elementos de todos los ámbitos
- M4_SCOPE_REGISTER: Elementos de registro (solamente)
- M4_SCOPE_BLOCK: Elementos de bloque (solamente)
- M4_SCOPE_NODE: Elementos de nodo (solamente)
- M4_TYPE_ALL: Todos los tipos de elemento

- M4_TYPE_METHOD: Elementos de tipo método
- M4_TYPE_PROPERTY: Elementos de tipo propiedad
- M4_TYPE_FIELD: Elementos de tipo campo
- M4_TYPE_CONCEPT: Elementos de tipo concepto

Puede aplicar uno o más criterios para el filtro a través del uso del operador booleano OR.

EJEMPLO

Para recuperar sólo los métodos cuyo ámbito sea Bloque del nodo llamado minodo:

```
GetNodeNumberOfItems( "minodo", M4_SCOPE_BLOCK + M4_TYPE_METHOD )
```

GetNodeIndex (Nodo)

Esta función devuelve el índice asociado a un nodo.

EJEMPLO DE SINTAXIS

```
GetNodeIndex (
   VALUE VARIABLE STRING Node
)
```

GetNodeItemId (Nodo, Posición, [Filtro])

Esta función devuelve el identificador del elemento del nodo que está en la posición indicada del filtro definido que se va a aplicar.

EJEMPLO DE SINTAXIS

```
GetNodeItemId (
   VALUE VARIABLE STRING Node,
   VALUE NUMBER Position,
   VALUE NUMBER Filter
)
```

GetNodeItemType (Nodo, Elemento, [Filtro]) o (Nodo, Elemento)

Esta función devuelve el tipo del elemento. Se le pueden indicar valores de dos formas:

- Identificador del nodo, posición y filtro del elemento
- Identificador del nodo y del elemento

Los argumentos correspondientes a esta función son los siguientes:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo.

  Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemType (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

Devuelve los siguientes posibles valores: Método, concepto, campo y propiedad.

Se puede utilizar macros que representen estos valores:

M4_TYPE_METHOD

M4_TYPE_PROPERTY

M4_TYPE_FIELD

M4_TYPE_CONCEPT

GetNodeItemScope (Nodo, Elemento, [Filtro]) o (Nodo, IDElemento)

Esta función devuelve el ámbito de un elemento determinado. Sus argumentos son los siguientes:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemScope (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

Devuelve los siguientes posibles valores: nodo, bloque y registro.

Se puede utilizar macros que representen estos valores:

M4_SCOPE_NODE

M4_SCOPE_BLOCK

M4_SCOPE_REGISTER

GetNodeItemM4Type (Nodo, Pos, [Filtro]) o (Nodo, IDElemento)

Esta función devuelve el tipo básico de un elemento de un nodo. Sus argumentos son los siguientes:

- **Nodo**: contiene el nombre del nodo que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:

  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia..

EJEMPLO DE SINTAXIS

```
GetNodeItemM4Type (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

Devuelve los siguientes posibles valores de tipos Meta4:

0: nulo

1: cadena fija

2: cadena variable

3: largo

4: fecha

5: timestamp

6: número

7: variante

8: moneda

9: número variante

10: blob

11: cadena binaria

12: tipo de hora

GetNodeItemPrecision (Nodo, Pos, [Filtro]) o (Nodo, IDElemento)

Esta función devuelve la precisión de un elemento. Sus argumentos son los siguientes:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.

- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

Esta función tiene diferentes aplicaciones, por ejemplo:

- Si el elemento es alfanumérico, permite saber el número de caracteres que puede tener el elemento.
- Si el elemento es numérico, la precisión, junto con la escala, permite saber el número de decimales con los que se va a guardar ese número en la base de datos.

EJEMPLO DE SINTAXIS

```
GetNodeItemPrecision (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

GetNodeItemPrecisionEx (Precision, Nodo, Pos, [Filtro]) o (Precision, Nodo, IDElemento)

Esta función devuelve por referencia la precisión de un elemento. Devuelve como resultado la constante M4_TRUE si se ejecuta correctamente, o M4_FALSE si se produce algún error.

Los argumentos correspondientes a esta función son los siguientes:

- **Precision**: indica la precisión del elemento especificado. La precisión se puede utilizar para diferentes fines.
  - Si el elemento es alfanumérico, esta función permite saber el número de caracteres que puede tener el elemento.
  - Si el elemento es numérico, la precisión, junto con la escala, permite saber el número de decimales con los que se va a guardar ese número en la base de datos.
- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.

- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemPrecisionEx (
   REFERENCE NUMBER Prec,
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

GetNodeItemIndex (Nodo, Pos, [Filtro]) o (Nodo, IDElemento)

Esta función devuelve el índice correspondiente a un elemento. En esta función se entiende por índice el ordinal que posee cada uno de los elementos en su nodo según el filtro aplicado.

Los argumentos correspondientes a esta función:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.

  Filtro: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemIndex (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
```

```
)
```

Si no existe el elemento en el nodo, la función devuelve un mensaje de aviso.

GetNodeItemScale (Nodo, Pos, [Filtro]) o (Nodo, IDElemento)

Esta función devuelve la escala correspondiente a un elemento. Este valor es muy importante y se le puede dar diferentes usos, por ejemplo, se puede utilizar cuando se tenga un elemento numérico, ya que la escala y la precisión permiten saber el número de decimales con los que se va a guardar ese número en la base de datos.

Los argumentos correspondientes a esta función son:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.

  [Filtro]: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemScale (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

ExistNodeItemIndex (Nodo, Posción, Filtro) o (Nodo, IDElemento)

Esta función devuelve el índice correspondiente a un elemento si existe. En esta función se entiende por índice el ordinal que posee cada uno de los elementos en su nodo según el filtro aplicado.

Los argumentos correspondientes a esta función:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **Filtro**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
ExistNodeItemIndex (
VALUE VARIABLE STRING Node,
VALUE VARIANT Item,
[VALUE NUMBER Filter]
)
```

No se genera ningún tipo de aviso si no existe el elemento.

GetNodeItemInternalType (Nodo, Pos, [Filtro]) o (Nodo, IDElemento)

Esta función permite conocer el tipo interno de un determinado elemento, como por ejemplo de Fecha de inicio, Fecha de fin, etc. Sus argumentos son los siguientes:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.

  [Filtro]: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemInternalType (
```

```
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

GetNodeItemReadObjectAlias (Nodo, Pos, Filtro) o (Nodo, IDElemento)

Esta función permite conocer el alias del objeto de lectura de un elememento. Sus argumentos son los siguientes:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **Filtro**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemReadObjectAlias (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

GetNodeItemWriteObjectAlias (Nodo, Pos, [Filtro]) o (Nodo, IDElemento)

Esta función permite conocer el alias del objeto de escritura de un elemento.

Sus argumentos son los siguientes:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:

  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemWriteObjectAlias (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

GetNodeItemReadFieldId (Nodo, Pos, [Filtro]) o (Nodo, IDElemento)

Esta función permite conocer el ID del campo de lectura de un elemento. Sus argumentos son los siguientes:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **Filtro**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemReadFieldId (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

GetNodeItemWriteFieldId (Nodo, Pos, [Filtro]) o (Nodo, IDElemento)

Esta función permite conocer el ID del campo de escritura de un elememento. Sus argumentos son los siguientes:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemWritefieldId (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

GetNodeItemReadObjectId (Nodo, Pos, [Filtro]) o (Nodo, IDElemento)

Esta función permite conocer el ID del objeto de lectura de un elemento. Sus argumentos son los siguientes:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro.
  - Identificador del elemento al que se hace referencia.
- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemReadObjectId (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

GetNodeItemWriteObjectId (Nodo, Pos, [Filtro]) o (Nodo, IDElemento)

Esta función permite conocer el ID del objeto de escritura de un elemento. Sus argumentos son los siguientes:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

Ejemplo DE SINTAXIS

```
GetNodeItemWriteObjectId (
   VALUE VARIABLE STRING Node,
   REFERENCE VARIANT Item,
   [VALUE NUMBER Filter]
)
```

GetNodeItemArgumentPos (Nodo, Pos, [Filtro], IDArg) o (Nodo, IDElemento, IDArg)

Esta función permite conocer la posición de un argumento correspondiente a un elemento. Sus argumentos son los siguientes:

- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Elemento**: este segundo argumento puede tomar uno de dos valores posibles:
  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.
- **IDArg**: es el identificador del argumento del que se pide la posición. Este argumento es del elemento.

EJEMPLO DE SINTAXIS

```
GetNodeItemArgumentPos (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter,]
   VALUE VARIABLE STRING ArgId
)
```

GetNodeItemArgumentNumber (NúmeroFijo, HasVarArg, NúmeroRef, Nodo, Posición, [Filtro]) o (NúmeroFijo, HasVarArg, NúmeroRef, Nodo, IDElemento)

Esta función permite conocer el número de argumentos del elemento. Los argumentos propios de la función son los siguientes:

- **NúmeroFijo**: parámetro de salida que contiene el número de argumentos fijos de los que se consta.
- **HasVarArg** :parámetro de salida que indica si el elemento tiene un número variable de argumentos.
- **NúmeroRef**: parámetro de salida que indica el número de argumentos de entrada y salida.
- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro.
- **IDElemento**: Identificador del elemento al que se hace referencia.

- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemArgumentPos (
   VALUE VARIABLE STRING Node,
   VALUE VARIANT Item,
   [VALUE NUMBER Filter,]
   VALUE VARIABLE STRING ArgId
)
```

GetNodeItemArgument (IdArg, TipoArg, Prec, IsRefArg, ArgPos, Nodo, Posición, [Filtro]) o (IdArg, TipoArg, Prec, IsRefArg, ArgPos, Nodo, Posición, IDElemento)

Esta función devuelve la información sobre el argumento ubicado en la posición especificada en el argumento *ArgPos* del elemento.

Los argumentos correspondientes a esta función son los siguientes:

- **IDArg**: parámetro de salida que contiene el ID del argumento.
- **TipoArg**: parámetro de salida que indica el tipo correspondiente al argumento.
- **Prec**: parámetro de salida que corresponde a la precisión del elemento.
- **IDRefArg**: parámetro de salida que indica si el argumento es de entrada o salida.
- **ArgPos**: indica la posición del argumento indicado como argumento.
- **Nodo**: contiene el identificador del nodo o el indice que se va a utilizar.
- **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro.
- **IDElemento**: Identificador del elemento al que se hace referencia.
- **[Filtro]**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.

EJEMPLO DE SINTAXIS

```
GetNodeItemArgumentPos (
   VALUE VARIABLE STRING Node,
```

```
    VALUE VARIANT Item,
    [VALUE NUMBER Filter,]
    VALUE VARIABLE STRING ArgId
)
```

GetNodeItemData([M4Obj], Nodo, Posición, Filtro, TipoDato) o ([M4Obj], Nodo, IDElemento, TipoDato)

Esta función devuelve la información sobre el tipo de metadato solicitado.

La sintaxis admitida es:

```
GetNodeItemData([M4Obj], Nodo, Pos, Filtro, TipoDato)
GetNodeItemData([M4Obj], Nodo, IDElemento, TipoDato)
```

Los argumentos correspondientes a esta función son los siguientes:

- **M4Obj:** contiene el identificador de la instancia de nivel 2 de Meta4Object que se va a utilizar. Este argumento es opcional y si no se especifica la función opera sobre el Meta4Object actual.
- **Nodo**: contiene el identificador del nodo que se va a utilizar, también se puede utilizar el indice de nodo.
- **Elemento**

  Este segundo argumento puede tomar uno de dos valores posibles:

  - **Posición**: Posición del elemento al que se hace referencia, en cuyo caso es necesario que se indique un filtro en el siguiente argumento.
  - **IDElemento**: Identificador del elemento al que se hace referencia.
- **Filtro**: contiene el filtro que se va a aplicar sobre el nodo. Este argumento es obligatorio únicamente cuando se indique la posición del elemento al que se hace referencia.
- **TipoDato**: indica cual es el tipo de metadatos a solictar. Los tipos de datos admitidos vienen dados por las siguientes macros que no deben ir entre comillas:

  M4_ITEM_TYPE

  M4_ITEM_SCOPE

  M4_ITEM_M4TYPE

  M4_ITEM_PRECISION

  M4_ITEM_INDEX

  M4_ITEM_SCALE

  M4_ITEM_INTERNAL_TYPE

  M4_ITEM_READ_OBJECT_ALIAS

M4_ITEM_WRITE_OBJECT_ALIAS
M4_ITEM_READ_FIELD_ID
M4_ITEM_WRITE_FIELD_ID
M4_ITEM_READ_OBJECT_ID
M4_ITEM_WRITE_OBJECT_ID
M4_ITEM_SLICE_TOTALIZE
M4_ITEM_NAME
M4_ITEM_ORDER
M4_ITEM_SYNONYM
M4_ITEM_ID
M4_ITEM_NUMBER
M4_ITEM_IS_PK

EJEMPLO DE SINTAXIS

Devolver el tipo de elemento de un elemento determinado:

```
...
type = GetNodeItemType( "MI_NODO", "MI_ELEMENTO")
type = GetNodeItemData( "MI_NODO", "MI_ELEMENTO", M4_ITEM_TYPE )
type = GetNodeItemData( "MI_M4OBJ", "MI_NODO", "MI_ELEMENTO",
M4_ITEM_TYPE )
...
```

Funciones de nómina

Las funciones que se engloban dentro de este apartado, ofrecen facilidades específicas a la hora de realizar cálculos de nómina.

YTDSearch (Meta4Object, Nodo, FechaInicio, FechaFin, FechaEntrada, FechaPago, PagoFrec, TipoPago, TipoBúsqueda, RevBeh, SelPays, TotOp)

Esta función calcula el acumulado de una estructura de nodo de amplias dimensiones que esté cargada en memoria y que conste de todos los datos correspondientes al acumulado.

La función va leyendo la información nodo a nodo, rellenando varias estructuras de nodo pequeñas con datos que cumplan un criterio específico.

Esta función no está implementada para llamadas de nivel 2, es decir, no es posible tener acceso a otros Meta4Objects distintos al actual.

EJEMPLO DE SINTAXIS

```
YTDSearch (
   VALUE VARIABLE STRING Meta4Object, VALUE VARIABLE STRING Node,
   VALUE DATE AND TIME StartDate, VALUE DATE AND TIME EndDate,
   VALUE DATE AND TIME ImputeDate, VALUE DATE AND TIME PayDate,
   VALUE VARIABLE STRING PayFrequency, VALUE VARIABLE STRING
   PayType, VALUE NUMBER SearchType, VALUE NUMBER RevBehaviour,
   VALUE NUMBER SelPays, VALUE NUMBER TotOperation
)
```

YTDSearchFiltered (Meta4Object, Nodo, FechaInicio, FechaFin, FechaEntrada, FechaPago, PagoFrec, TipoPago, TipoBúsqueda, RevBeh, SelPays, TotOp, [IDElemento, Valor, Función])

Esta función filtra las búsquedas en la estructura de nodo origen, por criterios de fecha, tales como periodo. Además, calcula el acumulado de una estructura de nodo leyendo los nodos de uno en uno.

Se puede utilizar con condiciones adicionales sobre el nodo, seleccionando aquellos registros que están en el bloque activo, del mismo modo que la función *FindRegister* (elemento, valor, función...).

También permite especificar condiciones adicionales que debe cumplir el registro para que se le tenga en cuenta cuando se ejecute el algoritmo *YTD Search*.

Especificación del filtro

El usuario puede indicar argumentos en grupos de tres, como en la función *FindRegister ( )*, mediante los que se especifica el *ID_Elemento*, o valor que se usa para comparar con la función en uso.

Al utilizar filtros, sólo se seleccionan los registros que cumplan las condiciones especificadas, que podrían ser hasta 20 distintas.

Ejecución con tramos

Si se va a llevar a cabo una ejecución con tramos, en el filtro que se utilice se puede indicar el valor de uno de ellos. De este modo, se harán varias llamadas a la función *YTDSearchFiltered*, una por tramo.

Por tanto, en el nodo destino se dispondrá de los resultados requeridos para ver diferentes valores de un elemento a lo largo del tiempo.

EJEMPLO

El elemento node.filter_item tiene dos tramos y se llama a la función del siguiente modo:

```
YTDSearchFiltered(..., "MY_ITEM", node.filter_item, EQUAL)
```

Se llamará a las funciones dos veces, como es normal en una ejecución con tramos. En el nodo destino se recogerán los resultados según el formato que se ha especificado.

Diferentes comportamientos con distintas llamadas

En la función existen varios argumentos opcionales [IDElemento, Valor, Función], en los que se debe especificar cómo se desea que aparezcan los resultados en el nodo destino.

Si se indica este argumento, se deberá hacer en la última posición. De lo contrario, sólo se guardará un registro.

El sistema de asignación de valores

Desde el Diseñador de modelo de datos, se puede gestionar un tipo especial de tablas: las tablas de valor. La función principal de estas tablas es la asignación de determinados valores asociados a conceptos.

Las tablas de valor son tablas fechadas, en las que todo se guarda como si se tratara de un histórico. Al igual que en los históricos, estas tablas tienen unas columnas fijas y otros que dependerán del nivel.

Los conceptos se pueden agrupar en función de la tipología del concepto, o la creación, función o ejecución del mismo. Por ejemplo: puede agrupar en Devengos todos los conceptos definidos como devengos genéricos que percibirá el empleado: salario base, dietas, plus de horas extras, etc.

Un dominio de metadatos (o DM) es, dentro de una jerarquía, el nivel superior en el que se agrupan los conceptos propios de aquellos módulos en los que es imprescindible la asignación de un valor para su ejecución.

La principal aplicación de los DM se encuentra relacionada con el cálculo de la nómina, aunque se extiende su funcionalidad a otros procesos dentro de la gestión de los recursos humanos.

Con idea de garantizar al máximo la integridad en el cálculo de la nómina y la confidencialidad en otros procesos de la gestión propia de los recursos humanos, la tecnología de Meta4 estructura un subsistema de seguridad basado en la creación de grupos de usuarios con permisos para modificar el valor asignado a los conceptos de nómina. Desde la tabla Grupos DMD, se van a definir los posibles grupos que va a tener asociado un DM, es decir, los permisos sobre los DM.

No olvide que al adscribir un DM a un grupo, el DM heredará todos los permisos concedidos al grupo.

El sistema de asignación de valores es un proceso que consta de una serie de funciones, cuyo orden lógico de ejecución es el siguiente:

- InitApplyValue
- SetPeriods
- SetDates
- ApplyTable
- TransferApplyValue

La transferencia se produce entre distintos Meta4Objects, pudiendo realizarse también en el mismo Meta4Object, aunque su fin no sea éste.

Con todas las funciones citadas, se realiza el proceso del sistema de asignación de valores. Todas las funciones, salvo *TransferApplyValue*, realizan la fase 1, y cuando por último se ejecuta ésta, se completa la fase 2.

En la fase 1 se utiliza un Meta4Object especial que consta de tablas, históricos, organigramas, información sobre el empleado, información sobre la empresa, etc. En esta fase, se leen tablas de valores, como el salario o las ausencias, que hayan ido incluyendo los usuarios. Esta información queda depositada en un nodo intermedio.

El pivotado de información consiste en cambiar la distribución de la información, de forma que en vez de aparecer en filas, los valores aparezcan en columnas.

En la fase 2 se toma esta información, se pivota, se transfiere a los nodos según la relación de dominio de metadatos y componentes del dominio de metadatos que exista y, por último, se deposita en un registro correspondiente al nodo de cálculo.

PrInitApplyValue (Id_DM, Grupo)

La tabla *DMD_components* contiene todos los componentes que se pueden aplicar utilizando esta función. Cualquier otro componente que exista y que no aparezca en esta lista, no se podrá aplicar con esta función.

Con esta función se indica con qué dominio de metadatos y con qué grupo DM se va a trabajar, y se asignan valores en la tabla. Posteriormente, se fuerza la carga de la tabla. Si el filtro es acorde con las fechas, la tabla queda filtrada por el grupo DM indicado.

El grupo DM es una agrupación de conceptos dentro del dominio de metadatos.

Los argumentos corrrespondientes a esta función son el identificador de la tabla DM que va a estar formada por una serie de conceptos, y el identificador del grupo de conceptos que forman parte del DM.

EJEMPLO DE SINTAXIS

```
prInitApplyValue (
   VALUE VARIABLE STRING IdDMD, VALUE VARIABLE STRING Group
)
```

PrSetPeriods (FechaInicio, FechaFin, FechaEjecución, ModoTramo)

Esta función permite indicar las fechas de ejecución del sistema de asignación de valores.

Los argumentos que recibe son las fechas de inicio y fin del tramo, la fecha de ejecución o corte y el modo de tramo que se va a utilizar.

EJEMPLO DE SINTAXIS

```
prSetPeriods (
   VALUE DATE AND TIME StartDate, VALUE DATE AND TIME EndDate,
   VALUE DATE AND TIME WorkDate, VALUE NUMBER SliceMode
)
```

PrSetDates (FechaInicio, FechaFin, [FechaEjecución])

Esta función permite indicar las fechas de ejecución (fechas de inicio y fin del tramo) para un individuo concreto cuya nómina se quiere calcular, así como la fecha de corte o de ejecución (esta última es un argumento opcional)..

La función se puede utilizar para filtrar tablas. Se utilizan las fechas en el nodo intermedio para determinar el periodo con el que se va a trabajar.

EJEMPLO DE SINTAXIS

```
prSetDates (
   VALUE DATE AND TIME StartDate, VALUE DATE AND TIME EndDate,
   [VALUE DATE AND TIME RunDate]
)
```

ApplyTable (NodoValores, NodoHistórico, NodoOrganización, [HistóricoCP, Valor sCP], iNumCP)

Esta función tiene como ámbito de aplicación las tablas, y ofrece diversos modos de ejecución:

– Sólo se indica el valor de las tablas de valores que son aplicables por fechas, DM, etc.

– Se indica también la tabla de históricos. Ésta aplica valores según las fechas del histórico. Para ello, se utiliza la relación de claves primarias, ya que de este modo en la tabla de históricos un elemento tendrá un valor igual al de la tabla de valores.

  Por ejemplo, el valor del elemento "departamento" de la tabla de histórico tiene que ser igual al valor del elemento "Pertenece al dpto" de la tabla de valores.

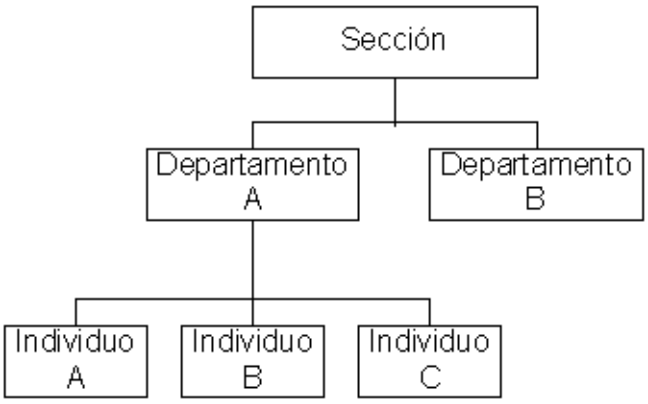
  Si se cumple esta condición, siempre comparando valores, significa que el registro de la tabla de valores es aplicable.

– La última posibilidad es por organigrama; en el Meta4Object del sistema de asignación de valores hay un nodo especial que contiene el organigrama, por ejemplo, un organigrama de la empresa.

  El sistema de asignación de valores actúa en función de este organigrama, de tal forma que se conecta a un nodo del organigrama, en el que intenta aplicar el valor en orden ascendente. A la hora de aplicar valores, tienen preferencia aquellos elementos más inferiores en la jerarquía, ascendiendo hasta el nodo raíz.

EJEMPLO

Imagine que tiene el siguiente organigrama:



El objetivo de este ejemplo es cambiar el salario base del individuo A que trabaja en el departamento A correspondiente a una sección.

Para llevar a cabo esta operación, el proceso necesario es el siguiente:

- Acceder a la tabla de históricos y comprobar que el individuo A existe, al igual que sucede en el organigrama de la organización.
- Acceder a la tabla de valores y ver el salario que tiene asignado el individuo A. Aquí se pueden dar dos situaciones:
  - El individuo A existe en la tabla de valores, por lo que se procede a cambiar el salario base sustituyendo el valor existente por el nuevo.

- El individuo A no existe en la tabla de valores, por lo que habría que ver en el organigrama de la organización el elemento del que depende, en este caso el Departamento A, y acceder de nuevo a la tabla de valores para consultar el salario que le corresponde de forma general a este departamento. En caso de no existir tampoco en la tabla de valores, hay que encontrar un elemento en el organigrama del que dependa el Individuo A que tenga su correspondencia en la tabla de valores, momento en el cual se procederá a cambiar el salario base por el actual.

Los argumentos correspondientes a esta función son los siguientes:

- NodoValores

  Indica el nodo de la tabla de valores.

- NodoHistórico

  Indica el nodo de la tabla de históricos. Puede ser ser una cadena vacía si no se están utilizando nodos históricos y organigramas.

- NodoOrganización

  Indica el nodo de la tabla de organigrama. Puede ser ser una cadena vacía si no se están utilizando nodos históricos y organigramas.

- HistóricoCP

  Indica el nombre de un elemento en la tabla de históricos.

- Valor sCP

  Indica el nombre de un elemento en la tabla de valores.

- iNumCP

  Indica el número de pares de claves primarias que se indican.

EJEMPLO DE SINTAXIS

```
ApplyTable (
VALUE VARIABLE STRING ValuesNode, VALUE VARIABLE STRING
HistoricNode, VALUE VARIABLE STRING OrgNode, [VALUE VARIABLE
STRING HistoricPK,] [VALUE VARIABLE STRING ValuesPK,] VALUE
NUMBER iNumPK
)
```

TransferApplyValue (Host_Meta4Object, id, Host_nodo_id, [DMD, MonedaDestino, FechaIntercambio, TipoIntercambio])

Esta función recibe el nombre de fase 2 del sistema de asignación de valores. Puede ejecutarse con o sin tramos, dependiendo de cómo se esté ejecutando la nómina. Se puede asignar prioridades a cada uno de los elementos.

EJEMPLO DE SINTAXIS

```
TransferApplyValue (
VALUE VARIABLE STRING HostM4ObjetId, VALUE VARIABLE STRING
HostNodeId, [VALUE VARIABLE STRING DMD,] [VALUE VARIABLE STRING
DestCurrency,] [VALUE DATE ExchangeDate,] [VALUE VARIABLE STRING
ExchangeType]
)
```

CreateOrgChart ( )

Esta función permite preparar un determinado organigrama para trabajar con él. De esta forma, es más fácil comprobar que no existen ciclos y trabajar con tramos.

Esta función se tiene que ejecutar sobre el nodo que se especifique como correspondiente al del organigrama.

Funciones de moneda

El Meta4Object Engine ofrece la posibilidad de trabajar con distintos tipos de moneda de una forma dinámica y transparente para el usuario.

El funcionamiento de la multimoneda se basa en tres aspectos:

- Meta4Object de cambio de moneda

  Es el repositorio de los tipos de cambio entre todas las monedas. El diseño de este Meta4Object permite almacenar los datos como un histórico de los distintos cambios que se han producido.

  Por tanto, un cambio de una moneda a otra se define por los identificadores de la moneda origen y destino así como por su fecha de cambio.

  El Meta4Object de cambio de moneda gestiona también el uso de monedas intermedias y monedas raíz, y tiene los métodos necesarios para realizar los cambios.

- Elementos de tipo moneda

  Existe un tipo de elemento moneda que se reconoce como elemento especial. Los elementos de tipo moneda permiten conectar campos como la fecha de cambio o el tipo de moneda relativo al elemento mediante elementos auxiliares.

  Cuando se convierten registros de una moneda a otra, los elementos de tipo moneda se actualizan junto con todos sus campos auxiliares.

- Funciones LN4 para el cambio de monedas

  Permiten convertir valores numéricos y registros entre distintas monedas.

Meta4Object de cambio de moneda

Todo el soporte de multimoneda está basado en el Meta4Object de cambio de moneda llamado *Exchange_Rates*.

Básicamente, este Meta4Object permite definir un histórico para cada tipo de moneda con la siguiente información:

- Número de decimales significativos de la moneda
- Moneda raíz para una determinada moneda
- Tipo de conversión al resto de monedas para una fecha de cambio y una fuente de tipo de cambio concretos
- Monedas auxiliares (intermedias) para realizar las conversiones que no son directas

La estructura del Meta4Object de moneda se describe en los siguientes apartados:

Nodo Currency

El nodo *Currency* corresponde al nodo raíz. En él se define la información relativa a cada moneda.

Los campos que identifican este nodo son los siguientes:

- ID_Currency

  Contiene el identificador de la moneda.

- NM_Currency

  Identifica el nombre de la moneda.

- Dec_Nb

  Contiene el número de decimales significativos.

- Id_Root_Currency

Identifica la moneda raíz (véase nota).

Cuando una moneda tiene una moneda raíz con fecha de inicio anterior a la fecha de cambio, significa que depende de ésta para realizar cualquier cambio. Por ejemplo, si la moneda PES tiene como raíz EUR a partir de 1999, la conversión de PES a cualquier moneda y viceversa se realizará a través de EUR siempre que la fecha de cambio sea posterior a 1999.

- St_Start_Root_Currency

  Identifica la fecha de inicio de la moneda raíz (véase nota).
- CurGetDecimals

  Corresponde a un método del Meta4Object.
- CurGetExchange

  Corresponde a un método del Meta4Object.
- CurGetExchangeRounded

  Corresponde a un método del Meta4Object.

Nodo Rates

Este nodo está conectado por un tipo de conexión Registro-Bloque (RB) con el nodo *Currency*. Este nodo contiene los cambios directos entre la moneda origen y todas las demás.

Los campos que identifican este nodo son los siguientes:

- ID_Currency

  Contiene el identificador de la moneda origen. Está conectado con *Currency*.
- ID_Refer_Cur

  Contiene el identificador de la moneda destino.
- Ex_type

  Identifica la clase de cambio.
- Dt_Start

  Almacena la fecha de inicio del cambio.
- Dt_End

  Guarda la fecha de fin del cambio.
- Rate

  Contiene el tipo de cambio, que es el factor multiplicador que se utiliza para pasar de la moneda origen a la moneda destino.

La clave primaria de la tabla del Meta4Object está compuesta por (ID_Currency, ID_Refer_Cur, Ex_type y Dt_Start). Por tanto, siempre que se acceda a un cambio directo, es necesario especificar la moneda origen, la moneda destino, la clase de cambio y la fecha de cambio.

La clase de cambio (*Ex_type*) permite tener distintos tipos de cambio para una moneda en una fecha específica. Esto cobra importancia cuando se cuenta con varias fuentes de financiación, por ejemplo, las cotizaciones de las monedas en diferentes mercados de valores.

Nodo Aux_Cur_Ex

Este nodo está conectado por un tipo de conexión RB con *Currency*, y en él se definen los cambios indirectos entre la moneda origen y la moneda destino.

Los campos que identifican a este nodo son:

- **ID_Currency**: contiene el identificador de la moneda origen. Está conectado con *Currency*.
- **ID_Currency_Des**: contiene el identificador de la moneda destino.
- **ID_Curren_Iterm**: contiene el identificador de la moneda intermedia.
- **Dt_Start**: almacena la fecha de inicio del cambio.
- **Dt_End**: guarda la fecha de fin del cambio.

Elementos de tipo moneda

Si un elemento es de tipo moneda, puede tener asociados mediante elementos auxiliares los siguientes campos (por orden):

- Moneda (*ID_Currency*) del elemento (obligatorio).
- Fecha de cambio del elemento (opcional).
- Tipo de cambio (*Ex_type*) del elemento (opcional).

Las ventajas de la utilización de elementos de tipo moneda son las siguientes

- El campo moneda contiene información que le identifica claramente. Es posible tener en el mismo registro varios elementos moneda con identificadores de moneda, fechas de cambio y tipos de cambio distintos.
- Identifica qué campos de una estructura de nodo contienen valores moneda. Esto es importante para el manejo y presentación de los Meta4Objects.
- Las funciones *CurExchangeRecord* y *CurExchangeRecordRounded* cambian los elementos moneda a la moneda especificada y actualizan sus campos auxiliares de manera automática.

Funciones LN4 para el cambio de monedas

Sólo se debe llamar a las funciones *CurGetExchange*, *CurGetDecimals* y *CurGetExchangeRounded* a través del Meta4Object de cambio de moneda denominado *Exchange_Rates*. Si se utilizan desde otro Meta4Object, la llamada debe hacerse por nivel 2.

A las funciones *CurExchangeRecord*, *CurGetExchangeCurrency*, *CurExchangeRecordRounded* y *CurGetCurrencyExchangeRounded*se las puede llamar directamente desde cualquier Meta4Object, ya que se encargarse internamente de manejar las llamadas de nivel 2 para acceder al Meta4Object.

CurExchangeRecord (MonedaDestino, bNuevoRegistro, [FechaCambio], [TipoCambio])

Esta función cambia los valores de todos los elementos de tipo moneda de un registro a la moneda deseada, teniendo en cuenta para ello el valor del argumento *MonedaDestino*, que indicará la moneda destino. Si se especifican los argumentos opcionales, *FechaCambio* y *TipoCambio*, estos tienen prioridad sobre los elementos auxiliares definidos para indicar la fecha y el tipo de cambio de la moneda.

Los argumentos correspondientes a este elemento son:

- **MonedaDestino**: identifica la moneda destino.
- **BNuevoRegistro**: es un indicador que especifica si se utiliza o no un nuevo registro.
- **[FechaCambio]**: argumento opcional que proporciona la fecha de cambio.
- **[TipoCambio]**: argumento opcional que proporciona el tipo de cambio.

EJEMPLO DE SINTAXIS

```
curExchangeRecord (
   VALUE CURRENCY DestCurrency,
   VALUE NUMBER bNewRecord,
   [VALUE CURRENCY ExchangeDate,]
   [VALUE NUMBER ExchangeType]
)
```

CurGetExchange (MonedaOrigen, MonedaDestino, FechaIntercambio, TipoIntercambio, Valor)

Esta función devuelve el valor final resultante del cambio de una moneda a otra en una fecha determinada. La llamada a esta función debe realizarse desde el Meta4Object *EXCHANGE_RATES*, que contiene los tipos de cambio.

Los argumentos que recibe son la moneda origen, la moneda destino, la fecha de cambio, el tipo de cambio y la cantidad de moneda que se va a convertir.

EJEMPLO DE SINTAXIS

```
curGetExchange (
   VALUE CURRENCY OriginCurrency,
   VALUE CURRENCY DestCurrency,
   VALUE DATE AND TIME ExchangeDate,
   VALUE VARIABLE STRING ExchangeType,
   VALUE CURRENCY Value
)
```

CurGetDecimals (MonedaOrigen)

Esta función devuelve los decimales asociados a la moneda activa. La llamada a esta función debe realizarse desde el Meta4Object *EXCHANGE_RATES*.

EJEMPLO DE SINTAXIS

```
curGetDecimals(
   VALUE CURRENCY Currency
)
```

CurGetExchangeCurrency (MonedaOrigen, MonedaDestino, FechaIntercambio, TipoIntercambio, Valor)

Esta función invoca al elemento *CurGetExchange* del Meta4Object *EXCHANGE_RATES* a través de una llamada interna de nivel 2, teniendo en cuenta una serie de valores que se indican como argumentos.

Dichos argumentos son la moneda origen, la moneda destino, la fecha de cambio, el tipo de cambio y la cantidad de moneda que se va a convertir.

EJEMPLO DE SINTAXIS

```
curGetExchangeCurrency(
   VALUE CURRENCY OriginCurrency,
   VALUE CURRENCY DestCurrency,
   VALUE DATE AND TIME ExchangeDate,
   VALUE VARIABLE STRING Exchangetype,
   VALUE NUMBER Value
)
```

CurExchangeRecordRounded (MonedaDestino, bNuevoRegistro, [FechaIntercambio], [TipoIntercambio])

Esta función, al igual que *curExchangeRecord*, redondea los valores que se han cambiado, al número de decimales asignado a la moneda destino.

La función tiene como argumentos la moneda destino y un indicador que especifica si se utiliza un nuevo registro, además de dos argumentos opcionales que proporcionan respectivamente la fecha de cambio y el tipo de cambio.

EJEMPLO DE SINTAXIS

```
curExchangeRecordRounded(
   VALUE CURRENCY DestCurrency,
   VALUE NUMBER bNewRecord,
   [VALUE CURRENCY ExchangeDate,]
   [VALUE NUMBER ExchangeType]
)
```

CurGetExchangeRounded (MonedaOrigen, MonedaDestino, FechaIntercambio, TipoIntercambio, Valor)

Esta función, al igual que *curGetExchange*, redondea los valores que se han cambiado al número de decimales indicado en la moneda de destino.

La llamada a esta función debe realizarse desde el Meta4Object *EXCHANGE_RATES*, que contiene el tipo de cambio que se debe usar.

Los argumentos correspondientes a esta función son la moneda origen, la moneda destino, la fecha de cambio, el tipo de cambio y la cantidad de moneda que se va a convertir.

EJEMPLO DE SINTAXIS

```
curGetExchangeRounded(
   VALUE CURRENCY OriginCurrency,
   VALUE CURRENCY DestCurrency,
   VALUE DATE AND TIME ExchangeDate,
   VALUE VARIABLE STRING ExchangeType,
   VALUE CURRENCY Value
)
```

CurGetCurrencyExchangeRounded(MonedaOrigen, MonedaDestino, FechaIntercambio, TipoIntercambio, Valor)

Esta función, al igual que *curGetExchangeCurrency*, redondea el valor que se ha cambiado al número de decimales asignado a la moneda de destino.

Los argumentos correspondientes a esta función son la moneda origen, la moneda destino, la fecha de cambio, el tipo de cambio, y la cantidad de moneda que se va a convertir.

EJEMPLO DE SINTAXIS

```
curGetExchangeCurrencyRounded(
   VALUE CURRENCY OriginCurrency,
   VALUE CURRENCY DestCurrency,
   VALUE DATE AND TIME ExchangeDate,
   VALUE VARIABLE STRING ExchangeType,
   VALUE CURRENCY Value
)
```

Ejemplos de funcionamiento del cambio de moneda

A continuación, se muestra una tabla que contiene una serie de valores ejemplo del Meta4Object *Exchange_Rates*, y posteriormente, se estudiarán distintos ejemplos de cambios de moneda.

Nodo Currency

ID_CURRENCY	DEC_NB	ID_ROOT_CURRENCY	DT_START ROOT_CURRENCY
EUR	5		
DOL	5		
PES	0	EUR	1/1/1999
MAR	2	EUR	1/1/1999
YEN	4		
CUR_EX	5		

Nodo Rates

ID_CURRENCY	ID_REFER_CUR	EX_TYPE	DT_START	DT_END	RATE
EUR	PES	1	1/1/1999	31/12/2001	168
EUR	MAR	1	1/1/1999	31/12/2001	2
EUR	DOL	1	1/1/1999	31/12/2001	1.176
EUR	YEN	1	1/1/1999	31/12/2001	3

ID_CURRENCY	ID_REFER_CUR	EX_TYPE	DT_START	DT_END	RATE
DOL	PES	1	1/1/1990	31/12/1998	145
DOL	MAR	1	1/1/1990	31/12/1998	1.8
DOL	EUR	1	1/1/1999	31/12/2001	0.85
DOL	YEN	1	1/1/1990	31/12/2001	2

ID_CURRENCY	ID_REFER_CUR	EX_TYPE	DT_START	DT_END	RATE
YEN	PES	1	1/1/1990	31/12/1998	70.5
YEN	MAR	1	1/1/1990	31/12/1998	0.9
YEN	EUR	1	1/1/1999	31/12/2001	0.425
YEN	DOL	1	1/1/1990	31/12/2001	0.5
YEN	CUR_EX	1	1/1/1990	31/12/2001	3

ID_CURRENCY	ID_REFER_CUR	EX_TYPE	DT_START	DT_END	RATE
PES	DOL	1	1/1/1990	31/12/1998	0.00689
PES	MAR	1	1/1/1990	31/12/1998	0.0124
PES	YEN	1	1/1/1999	31/12/1998	0.0138

ID_CURRENCY	ID_REFER_CUR	EX_TYPE	DT_START	DT_END	RATE
MAR	DOL	1	1/1/1990	31/12/1998	0.555
MAR	PES	1	1/1/1990	31/12/1998	80.5
MAR	YEN	1	1/1/1999	31/12/1998	1.111

ID_CURRENCY	ID_REFER_CUR	EX_TYPE	DT_START	DT_END	RATE
CUR_EX	YEN	1	1/1/1990	31/12/1998	0.333

Nodo Aux_Cur_Ex

ID_CURRENCY	ID_CURRENCY_DES	ID_CURREN_INTERM	DT_START	DT_END
CUR_EX	PES	YEN	1/1/1990	31/12/1999
CUR_EX	MAR	YEN	1/1/1990	31/12/1999

ID_CURRENCY	ID_CURRENCY_DES	ID_CURREN_INTERM	DT_START	DT_END
CUR_EX	DOL	YEN	1/1/1990	31/12/2001
CUR_EX	EUR	YEN	1/1/1999	31/12/2001

ID_CURRENCY	ID_CURRENCY_DES	ID_CURREN_INTERM	DT_START	DT_END
PES	CUR_EX	YEN	1/1/1990	31/12/1999

ID_CURRENCY	ID_CURRENCY_DES	ID_CURREN_INTERM	DT_START	DT_END
MAR	CUR_EX	YEN	1/1/1990	31/12/1999

ID_CURRENCY	ID_CURRENCY_DES	ID_CURREN_INTERM	DT_START	DT_END
EUR	CUR_EX	YEN	1/1/1999	31/12/2001

ID_CURRENCY	ID_CURRENCY_DES	ID_CURREN_INTERM	DT_START	DT_END
DOL	CUR_EX	YEN	1/1/1990	31/12/2001

A continuación, se muestran los resultados que se obtienen al realizar distintos cambios de moneda entre las tablas.

Los resultados se indican en una tabla de una columna y dos filas. En la primera fila se muestra el código LN4 utilizado y en la segunda el resultado obtenido de su ejecución.

ValueExchanged = CurGetExchange(YEN, DOL, 1/1/2030, 1, 30)
Cambio(Src=YEN, Dst=DOL, Date=1/1/2030, Type=1) ➔ 1 (Error) ValueExchanged = 30

ValueExchanged = CurGetExchange(DOL, MAR, 1/1/1996, 1, 30)
Cambio(Src=DOL, Dst=MAR, Date=1/1/1996, Type=1) ➔ 1.8 (Exito) ValueExchanged = 54

ValueExchanged = CurGetExchange(DOL, MAR, 10/10/1999, 1, 30)
Cambio(Src=DOL, Dst=MAR, Date=10/10/1999, Type=1) ➔ Cambio(DOL, EUR, 10/10/1999, 1) * Cambio(EUR, MAR, 10/10/1999, 1) ➔ ➔ 0.85 * 2 ➔ ➔ 1.7 (Exito) ValueExchanged = 51

ValueExchanged = CurGetExchange(PES, DOL, 10/10/1999, 1, 30)
Cambio(Src=PES, Dst=DOL, Date=10/10/1999, Type=1) ➔ CambioInverso (EUR, PES, 10/10/1999, 1) * Cambio(EUR, DOL, 10/10/1999, 1) ➔ ➔ 1/168 * 1.176 ➔ ➔ 0.007 (Exito) ValueExchanged = 0.21

ValueExchanged = CurGetExchange(PES, MAR, 10/10/1999, 1, 30)
Cambio(Src=PES, Dst=MAR, Date=10/10/1999, Type=1) ➔ CambioInverso (EUR, PES, 10/10/1999, 1) * Cambio(EUR, MAR, 10/10/1999, 1) ➔ ➔ 1/168 * 2 ➔ ➔ 0.0119 (Exito) ValueExchanged = 0.357

ValueExchanged = CurGetExchange(PES, MAR, 10/10/1995, 1, 30)
Cambio(Src=PES, Dst=MAR, Date=10/10/1995, Type=1) ➔ 0.0124 (Exito) ValueExchanged = 0.372

Funciones para estadísticas

DumpStatistics (NombreArchivo, Meta4Object, [Nivel])

Esta función vuelca en un archivo estadísticas del Meta4Object (número de bloques, registros, nodos, etc). Sus argumentos son:

- **NombreArchivo**: contiene el nombre de archivo destino en el que se van a guardar las estadísticas que se obtengan del Meta4Object.
- **Meta4Object**: corresponde al nombre del Meta4Object cuyas estadísticas se van a elaborar.
- **Nivel**: este argumento es opcional e indica el nivel de profundidad en Meta4Objects invocados por llamadas de nivel 2.

EJEMPLO DE SINTAXIS

```
DumpStatistics (
   VALUE FIXED STRING FileName,
   VALUE FIXED STRING Meta4Object,
   [VALUE NUMBER Level]
)
```

SttSetLabel (Etiqueta)

Esta función asigna una etiqueta a las estadísticas activas y todas sus dependencias.

SttPersistToFile (NombreArchivo)

Esta función graba todas las estadísticas del Meta4Object en el archivo especificado, siempre que las estadísticas estén activas (Nivel |= 0).

EJEMPLO DE SINTAXIS

```
sttPersistToFile (
   VALUE VARIABLE STRING FileName
)
```

Macros LN4

Las macros del lenguaje LN4 son una serie de constantes que han sido definidas internamente por Meta4 Software, S.A.

Estas macros tienen asignadas un modo determinado de actuación, y se invocan escribiendo directamente el nombre de la constante.

Existen macros de diverso tipo:

- Macros básicas
- Macros de búsqueda
- Macros para cuadros de diálogo
- Macros para gestionar botones en cuadros de diálogo
- Macros de funciones de totalización
- Macros para transacciones en base de datos
- Macros para manipulación de cadenas
- Macros de tipo y ámbito de elementos
- Macros de constantes CHR
- Macros de constantes de indirección de atributos (lectura)
- Macros de constantes de indirección de atributos (ejecución)
- Macros de autocarga
- Macros de constantes de log
- Macros de constantes de ordenación
- Macros de valores de modo de envío
- Macros de constantes de fecha
- Macros de tipos variantes
- Macros de políticas de compartición
- Macros de funciones de fecha (suma/diferencia)
- Macros matemáticas
- Macros de metadatos de elementos

Macros básicas

M4_TRUE

Esta macro simboliza el valor booleano true.

M4_FALSE

Esta macro simboliza el valor booleano FALSE.

M4_SHORT_DATE_FORMAT

Es el formato de fecha corto. A diferencia de la constante *M4_Long_Date_Format*, esta macro no añade segundos, minutos ni horas.

M4_LONG_DATE_FORMAT

Es el formato de fecha largo. A diferencia de la constante *M4_Short_Date_Format*, esta macro añade las horas, minutos y segundos.

M4_SUCCESS

Esta macro indica cuándo una función se ha ejecutado correctamente.

M4_ERROR

Esta macro indica cuándo una función no se ha ejecutado correctamente.

M4_ZERO

Esta macro simboliza el valor cero.

M4_POSITIVE

Esta macro simboliza un valor positivo.

M4_NEGATIVE

Esta macro simboliza un valor negativo.

Macros de búsqueda

M4_EQUAL

Esta macro permite buscar un registro en el que el valor que se busca sea igual a la cadena indicada.

M4_DISTINCT

Esta macro permite buscar un registro en el que el valor que se busca sea distinto a la cadena indicada.

M4_GREATER

Esta macro permite buscar un registro en el que el valor que se busca sea mayor que el indicado.

M4_SMALLER

Esta macro permite buscar un registro en el que el valor que se busca sea menor que el indicado.

M4_GREATER_OR_EQUAL

Esta macro permite buscar un registro en el que el valor que se busca sea mayor o igual que el indicado.

M4_SMALLER_OR_EQUAL

Esta macro permite buscar un registro en el que el valor que se busca sea menor o igual que el indicado.

M4_EQUAL_OR_NULL

Busca un registro cuyo valor que se sea igual que el indicado o nulo.

M4_DISTINCT_OR_NULL

Esta macro permite buscar un registro en el que el valor que se busca sea distinto que el indicado o nulo.

M4_GREATER_OR_NULL

Esta macro permite buscar un registro en el que el valor que se busca sea mayor que el indicado o nulo.

M4_SMALLER_OR_NULL

Esta macro permite buscar un registro en el que el valor que se busca sea menor que el indicado o nulo.

M4_GREATER_OR_EQUAL_OR_NULL

Esta macro permite buscar un registro en el que el valor que se busca sea mayor o igual que el indicado, o nulo.

M4_SMALLER_OR_EQUAL_OR_NULL

Esta macro permite buscar un registro en el que el valor que se busca sea menor o igual que el indicado, o nulo.

M4_CASE_EQUAL

Esta macro permite buscar un registro en el que el valor que se busca sea igual al indicado, distinguiendo entre mayúsculas y minúsculas.

M4_CASE_DISTINCT

Esta macro permite buscar un registro en el que el valor que se busca sea distinto al indicado, distinguiendo entre mayúsculas y minúsculas.

M4_CASE_GREATER

Esta macro busca un registro en el que el valor que se busca sea mayor que el indicado, distinguiendo entre mayúsculas y minúsculas.

M4_CASE_SMALLER

Esta macro busca un registro en el que el valor que se busca sea menor que el indicado, distinguiendo entre mayúsculas y minúsculas.

M4_CASE_GREATER_OR_EQUAL

Esta macro permite buscar un registro en el que el valor que se busca sea mayor o igual que el indicado, distinguiendo entre mayúsculas y minúsculas.

M4_CASE_SMALLER_OR_EQUAL

Esta macro permite buscar un registro en el que el valor que se busca sea menor o igual que el indicado, distinguiendo entre mayúsculas y minúsculas.

M4_CASE_EQUAL_OR_NULL

Esta macro permite buscar un registro en el que el valor que se busca sea igual que el indicado o nulo, distinguiendo entre mayúsculas y minúsculas.

M4_CASE_DISTINCT_OR_NULL

Esta macro permite buscar un registro en el que el valor que se busca sea distinto que el indicada o nulo, distinguiendo entre mayúsculas y minúsculas.

M4_CASE_GREATER_OR_NULL

Esta macro busca un registro en el que el valor que se busca sea mayor que el indicado o nulo, distinguiendo entre mayúsculas y minúsculas.

M4_CASE_SMALLER_OR_NULL

Esta macro permite buscar un registro en el que el valor que se busca sea menor que el indicado o nulo, distinguiendo entre mayúsculas y minúsculas.

M4_CASE_GREATER_OR_EQUAL_OR_NULL

Permite buscar un registro en el que el valor que se busca sea mayor o igual que el indicado, o nulo, distinguiendo entre mayúsculas y minúsculas.

M4_CASE_SMALLER_OR_EQUAL_OR_NULL

Permite buscar un registro en el que el valor que se busca sea menor o igual que el indicado o nula, distinguiendo entre mayúsculas y minúsculas.

M4_REGULAR_EXPRESSION

Esta macro permite establecer una serie de requisitos que debe cumplir el valor del elemento. Por ejemplo, que el valor de un elemento empiece por una determinada cadena o un determinado carácter, etc.

M4_REGULAR_EXPRESSION_OR_NULL

Esta macro permite establecer una serie de requisitos que debe cumplir el valor del elemento, a no ser que se trate de un valor nulo. Los requisitos pueden ser, por ejemplo, que el valor del elemento empiece por una determinada cadena o un determinado carácter, etc.

M4_CASE_REGULAR_EXPRESSION

Esta macro permite establecer una serie de requisitos que debe cumplir el valor del elemento. Por ejemplo, que el valor del elemento empiece por una determinada cadena o un determinado carácter, etc., distinguiendo entre mayúsculas y minúsculas.

M4_CASE_REGULAR_EXPRESSION_OR_NULL

Esta macro permite establecer una serie de requisitos que debe cumplir el valor del elemento, a menos que se trate de un valor nulo. Los requisitos

pueden ser, por ejemplo, que el valor del elemento empiece por una determinada cadena o un determinado carácter, etc.,distinguiendo entre mayúsculas y minúsculas.

Macros para cuadros de diálogo

Las macros que a continuación se exponen se utilizan para indicar los botones que se desean que aparezcan en el cuadro de diálogo.

M4_BTN_OK

Esta macro indica que aparezca el botón Aceptar en el cuadro de diálogo que se muestra al usuario.

M4_BTN_OK_CANCEL

Esta macro indica que aparezcan los botones Aceptar y Cancelar en el cuadro de diálogo que se muestra al usuario.

M4_BTN_ABORT_RETRY_IGNORE

Esta macro indica que aparezcan los botones Abortar, Reintentar e Ignorar en el cuadro de diálogo que se muestra al usuario.

M4_BTN_YES_NO_CANCEL

Esta macro indica que aparezcan los botones Sí, No y Cancelar en el cuadro de diálogo que se muestra al usuario.

M4_BTN_YES_NO

Esa macro indica que aparezcan los botones Sí y No en el cuadro de diálogo que se muestra al usuario.

M4_BTN_RETRY_CANCEL

Esa macro indica que aparezcan los botones Reintentar y Cancelar en el cuadro de diálogo que se muestra al usuario.

Macros para gestionar botones en cuadros de diálogo

Las macros que a continuación se exponen se utilizan para saber el botón que el usuario ha pulsado de todos los que aparecen en el cuadro de diálogo.

M4_OK

Esta macro indica que el usuario ha pulsado el botón Aceptar situado en el cuadro de diálogo que se ha mostrado en pantalla.

M4_CANCEL

Esta macro indica que el usuario ha pulsado el botón Cancelar situado en el cuadro de diálogo que se ha mostrado en pantalla.

M4_ABORT

Esta macro indica que el usuario ha pulsado el botón Abortar situado en el cuadro de diálogo que se ha mostrado en pantalla.

M4_RETRY

Esta macro indica que el usuario ha pulsado el botón Reintentar situado en el cuadro de diálogo que se ha mostrado en pantalla.

M4_IGNORE

Esta macro indica que el usuario ha pulsado el botón Ignorar situado en el cuadro de diálogo que se ha mostrado en pantalla.

M4_YES

Esta macro indica que el usuario ha pulsado el botón Sí situado en el cuadro de diálogo que se ha mostrado en pantalla.

M4_NO

Esta macro indica que el usuario ha pulsado el botón No situado en el cuadro

de diálogo que se ha mostrado en pantalla.

Macros de funciones de totalización

Cuando se define un elemento, se le asigna una función. Esto es de gran utilidad, ya que cuando se totaliza y no se indica ninguna otra particularidad, se utilizaría esa función definida como función por defecto.

En este tipo de funciones, existen dos atributos clave:

- *SystotalSlice*: totaliza los tramos de un elemento y los totaliza.
- *SysTotalItem*: totaliza cada uno de los elementos en cada uno de sus registros y los totaliza.

A continuación, se detallan otras funciones definidas por las que se puede totalizar.

M4_TOTAL_COUNT

Esta macro simboliza la totalización del recuento.

M4_TOTAL_SUM

Esta macro simboliza la totalización de la suma.

M4_TOTAL_AVG

Esta macro simboliza la totalización de la media.

M4_TOTAL_MAX

Esta macro simboliza la totalización del máximo.

M4_TOTAL_MIN

Esta macro simboliza la totalización del mínimo.

M4_TOTAL_FIRST

Esta macro simboliza la totalización y considera el resultado como primer valor.

M4_TOTAL_LAST

Esta macro simboliza la totalización y considera el resultado como último valor.

Macros para transacciones en base de datos

En este apartado se reúnen macros que son de gran utilidad a la hora de realizar operaciones durante una transacción de base de datos iniciada previamente.

M4_COMMIT

Esta macro simboliza que los cambios realizados en una transacción de base de datos que se ha iniciado previamente se graban permanentemente.

M4_ROLLBACK

Esta macro simboliza que se desechan los cambios realizados en una transacción de base de datos que se ha iniciado previamente.

Por ejemplo, puede utilizarse cuando interese deshacer los cambios realizados desde el inicio de una transacción en caso de producrise un fallo en la ejecución de una función, o bien para detener una ejecución en un momento determinado por cualquier motivo.

M4_EXECUTE_POSTVALIDATION

Esta macro comprueba si las validaciones posteriores funcionan correctamente o no. Esta posibilidad es de gran utilidad para casos de importación.

Macros para manipulación de cadenas

En este apartado, se reúnen las macros utilizadas para eliminar espacios de una cadena de texto especificada. Estas macros se utilizan con la función *Trim (Cadena, Dirección)*.

M4_TRIM_LEFT

Esta macro indica que se eliminan los espacios que existan en el lado izquierdo de la cadena.

M4_TRIM_ALL

Esta macro indica que se eliminan los espacios que existan a ambos lados de la cadena.

M4_TRIM_RIGHT

Esta macro indica que se eliminan los espacios que existan en el lado derecho de la cadena.

M4_LOWERCASE

Esta macro indica que se convierte a minúsculas una cadena dada.

M4_UNCHANGED

Esta macro indica que una cadena se deja tal cual se escribió.

M4_UPPERCASE

Esta macro indica que se convierte a mayúsculas una cadena dada.

Macros de tipo y ámbito de elementos

M4_SCOPE_ALL

Con esta función se indica que el ámbito de un elemento puede ser el registro, el bloque y el nodo.

M4_SCOPE_REGISTER

Función que indica que el ámbito de un elemento va a ser de tipo registro.

M4_SCOPE_BLOCK

Con esta función se indica que el ámbito de un elemento va a ser de tipo bloque.

M4_SCOPE_NODE

Con esta función se indica que el ámbito de un elemento va a ser de tipo nodo.

M4_TYPE_ALL

Con esta función se indica que el tipo de elemento puede ser Método, Propiedad, Campo o Concepto.

M4_TYPE_METHOD

Con esta función se indica que el tipo de elemento va a ser Método.

M4_TYPE_PROPERTY

Con esta función se indica que el tipo de elemento va a ser Propiedad.

M4_TYPE_FIELD

Con esta función se indica que el tipo de elemento va a ser Campo.

M4_TYPE_CONCEPT

Con esta función se indica que el tipo de elemento va a ser Concepto.

Macros de constantes CHR

Son macros útiles cuando se quiere escribir a nivel de código un carácter, como puede ser un tabulador, comillas, nueva línea, etc.

M4_NEW_LINE

Esta macro se utiliza cuando se quiere escribir una nueva línea.

M4_TAB

Esta macro se utiliza cuando se quiere introducir en una cadena un tabulador.

M4_DOUBLE_QUOTE

Esta macro se utiliza cuando se quiere escribir en una cadena unas comillas.

M4_CR

Esta macro se utiliza cuando se quiere escribir una nueva línea y un retorno de carro.

Macros de constantes de indirección de atributos (lectura)

Estas macros se utilizan cuando se quiera leer un atributo concreto. Para cada uno de los atributos hay una macro específica.

Para más información sobre estas macros, consulte el apartado *Atributos y Atributos-Métodos*.

Este tipo de constantes se refieren a los atributos que se ejecutan por indirección. Por ejemplo, los correspondientes a ChannelAttribCall, engloban:

- M4_ATT_SYS_SLICE_NUMBER
- M4_ATT_SYS_START_DATE
- M4_ATT_SYS_END_DATE
- M4_ATT_SYS_FIRST_SLICE
- M4_ATT_SYS_LAST_SLICE
- M4_ATT_SYS_OLD_VALUE
- M4_ATT_SYS_BLOB_DESCRIPTION
- M4_ATT_SYS_BLOB_MASK
- M4_ATT_SYS_AUX_ITEM_ID

M4_ATT_SYS_SLICE_NUMBER

Se aplica a elementos e indica el número de tramos que contiene un elemento.

M4_ATT_SYS_START_DATE

Se aplica a tramos y corresponde a la fecha de inicio del tramo.

M4_ATT_SYS_END_DATE

Se aplica a tramos y corresponde a la fecha de fin del tramo.

M4_ATT_SYS_FIRST_SLICE

Se aplica a elementos y corresponde al índice del primer tramo del elemento.

M4_ATT_SYS_LAST_SLICE

Se aplica a elementos y corresponde al índice del último tramo del elemento.

M4_ATT_SYS_OLD_VALUE

Se aplica a elementos y tramos, e indica el valor antiguo del elemento.

M4_ATT_SYS_BLOB_DESCRIPTION

Se aplica a elementos y devuelve el nombre del elemento.

M4_ATT_SYS_BLOB_MASK

Se aplica a elementos, e indica la máscara BLOB del elemento.

M4_ATT_SYS_AUX_ITEM_ID

Se aplica a elementos, e indica el identificador del elemento auxiliar de un elemento.

Para más información sobre esta y otras macros, consulte el apartado *Atributos y Atributos-Métodos*.

Macros de constantes de indirección de atributos (ejecución)

Estas macros se utilizan cuando se quiere ejecutar por indirección un atributo concreto. Para cada uno de los atributos hay una macro concreta.

Por ejemplo, los correspondientes a *ChannelAttribValue*, engloban:

- M4_ATT_SYS_ADD_NEW_SLICE
- M4_ATT_SYS_CREATE_SLICE
- M4_ATT_SYS_SPLIT_SLICE
- M4_ATT_SYS_TOTALIZE_ITEMS
- M4_ATT_SYS_TOTALIZE_SLICES
- M4_ATT_SYS_CREATE_BLOB_FILES

M4_ATT_SYS_ADD_NEW_SLICE

Se aplica a elementos e indica que se ha añadido un nuevo tramo al elemento.

M4_ATT_SYS_CREATE_SLICE

Se aplica a elementos y tramos, e indica que se ha añadido un nuevo tramo al elemento y que se le ha asignado un valor.

M4_ATT_SYS_SPLIT_SLICE

Se aplica a tramos, e indica que el tramo se ha dividido en dos utilizando la fecha, manteniendo así el valor acorde al comportamiento.

M4_ATT_SYS_TOTALIZE_ITEMS

Se aplica a elementos, e indica que adquiere como valor la totalización de este elemento.

M4_ATT_SYS_TOTALIZE_SLICES

Se aplica a tramos, e indica que adquiere como valor la totalización de este elemento.

M4_ATT_SYS_CREATE_BLOB_FILE

Se aplica a elementos, e indica la creación de un archivo para el elemento en el directorio con esta extensión.

M4_ATT_SYS_SET_BLOB_DESCRIPTION

Se aplica a elementos, e indica la asignación de la descripción de un elemento.

Macros de autocarga

La carga automática es una característica de los Meta4Object, que permite que se carguen en memoria a medida que se vayan utilizando.

M4_AUTOLOAD_OFF

Esta macro significa que por defecto esta función no está activa y en caso de que se quiera utilizar, el usuario debe hacer la llamada.

M4_AUTOLOAD_BLOCK

Esta macro significa la carga completa de un bloque entero.

M4_AUTOLOAD_PRG

Esta macro indica que la propiedad *Autoload* es de tipo propagante, es decir, que se carga el bloque y sus hijos.

M4_AUTOLOAD_NODESAYS

La carga automática hace lo que el nodo tiene configurado. Cada vez que se hace uso de la carga automática, se consulta la configuración del nodo para ver el tipo de carga que tiene predefinido.

Macros de constantes de log

M4_ERRORLOG

Esta macro representa los mensajes de error.

M4_WARNINGLOG

Esta macro representa los avisos generados.

M4_DEBUGINFOLOG

Esta macro representa los procesos de depuración.

Macros de constantes de ordenación

Bajo este apartado se recogen todas las macros utilizadas conjuntamente con la función de ordenación *AddSortFilter ( )*, tales como:

- M4_ASCENDING
- M4_DESCENDING

M4_ASCENDING

Esta macro es un criterio de clasificación ascendente.

M4_DESCENDING

Esta macro es un criterio de clasificación descendente.

Macros de valores de modo de envío

M4_SEND_NO_BRANCH

Esta macro indica que un nodo ha sido marcado para enviar, pero no así sus hijos.

M4_SEND_BRANCH

Esta macro indica que se ha marcado un nodo y sus hijos para enviar.

M4_SEND_RESET

Esta macro indica que se ha deshabilitado un nodo y, por lo tanto, no se va a enviar.

Macros de constantes de fecha

M4_MINUS_INF

Esta macro es el límite establecido como fecha mínima inferior. Su valor es 01-01-1800.

M4_PLUS_INF

Esta macro es el límite establecido como fecha máxima superior. Su valor es 01-01-4000.

Macros de tipos variantes

El uso de estas macros corresponde a la función *GetVariantType*, en la cual se indica un valor, y la función utilizada dice si es una cadena, fecha o número.

Estas funciones son de gran utilidad en aquellas ocasiones en las que se pasa un argumento y se desconoce su tipo, en función del cual actuará de un modo u otro.

M4_VARIANT_TYPE_NULL

El tipo de variante utilizada en esta macro es nulo.

M4_VARIANT_TYPE_NUMBER

El tipo de variante utilizada en esta macro es número.

M4_VARIANT_TYPE_STRING

El tipo de variante utilizada en esta macro es cadena.

M4_VARIANT_TYPE_DATE

El tipo de variante utilizada en esta macro es fecha.

Macros de políticas de compartición

Este tipo de macros se utilizan para las funciones siguientes:

- DefineInstance
- OpenChannel
- DefineInstanceEX
- OpenChannelEX

Estas macros disponen de diferentes tipos de compartición:

- M4_INSTANCE_NOT_SHARED
- M4_INSTANCE_LOCAL_SHARED
- M4_INSTANCE_GLOBAL_SHARED

M4_INSTANCE_NOT_SHARED

Esta macro indica que es una instancia no compartida.

M4_INSTANCE_LOCAL_SHARED

Esta macro indica que es una instancia compartida localmente.

M4_INSTANCE_GLOBAL_SHARED

Esta macro indica que es una instancia compartida globalmente.

Macros de funciones de fecha (suma/diferencia)

Este tipo de funciones indican:

- La unidad destino en la que se quiere obtener la diferencia entre dos fechas.
- El tipo de unidad con la que se va a incrementar una fecha.

M4_YEAR

Indica que la diferencia entre dos fechas o el incremento de una fecha se va a dar en años.

M4_MONTH

Indica que la diferencia entre dos fechas o el incremento de una fecha se va a dar en meses.

M4_DAY

Indica que la diferencia entre dos fechas o el incremento de una fecha se va a dar en días.

M4_WEEK

Indica que la diferencia entre dos fechas o el incremento de una fecha se va a dar en semanas completas.

M4_WEEKDAY

Indica que la diferencia entre dos fechas o el incremento de una fecha se va a dar en semanas.

Se entiende como semanas distintas un periodo que cubre dos días por ejemplo, perteneciendo un día a una semana y otro día a otra. Siempre es independiente del número de días.

M4_HOUR

Indica que la diferencia entre dos fechas o el incremento de una fecha se va a dar en horas.

M4_MINUTE

Indica que la diferencia entre dos fechas o el incremento de una fecha se va a dar en minutos.

M4_SECOND

Indica que la diferencia entre dos fechas o el incremento de una fecha se va a dar en segundos.

Macros matemáticas

M4_MATH_PI

Representa al valor de la constante *Pi* que es 3.1415

M4_MATH_E

Representa el valor de la constante neperiana *e* que es 2.71

Macros de metadatos de elementos

El uso de estas macros corresponde a la función GetNode ItemDataType, en la cual se indica un elemento concreto y una macro que describe un tipo de metadato cuyo valor se requiere.

Las macros de metadatos de elementos son:

M4_ITEM_TYPE

M4_ITEM_SCOPE

M4_ITEM_M4TYPE

M4_ITEM_PRECISION

M4_ITEM_INDEX

M4_ITEM_SCALE

M4_ITEM_INTERNAL_TYPE

M4_ITEM_READ_OBJECT_ALIAS

M4_ITEM_WRITE_OBJECT_ALIAS

M4_ITEM_READ_FIELD_ID

M4_ITEM_WRITE_FIELD_ID

M4_ITEM_READ_OBJECT_ID

M4_ITEM_WRITE_OBJECT_ID

M4_ITEM_SLICE_TOTALIZE

M4_ITEM_NAME

M4_ITEM_ORDER

M4_ITEM_SYNONYM

M4_ITEM_ID

M4_ITEM_NUMBER

M4_ITEM_TYPE

Con esta macro se indica que quiere recuperar el tipo del elemento, que puede ser método, propiedad, campo o concepto.

M4_ITEM_SCOPE

Con esta macro se indica que quiere recuperar el ámbito del elemento que puede ser registro, bloque o nodo.

M4_ITEM_M4TYPE

Con esta macro se indica que quiere recuperar el dato acerca del tipo M4, que puede ser nulo, cadena de longitud fija, cadena de longitud variable, campo long, fecha, fecha y hora, número, variant, moneda, variant numérico, fichero, campo long binario o hora.

M4_ITEM_PRECISION

Con esta macro se indica que quiere recuperar la precisión del elemento.

M4_ITEM_INDEX

Con esta macro se indica que quiere recuperar el indice del elemento.

M4_ITEM_SCALE

Con esta macro se indica que quiere recuperar la escala del elemento.

M4_ITEM_INTERNAL_TYPE

Con esta macro se indica que quiere recuperar el tipo interno del elemento.

M4_ITEM_READ_OBJECT_ALIAS

Con esta macro se indica que quiere recuperar el alias del objeto de lectura del elemento.

M4_ITEM_WRITE_OBJECT_ALIAS

Con esta macro se indica que quiere recuperar el alias del objeto de escritura del elemento.

M4_ITEM_READ_FIELD_ID

Con esta macro se indica que quiere recuperar el identificador del campo de lectura del elemento.

M4_ITEM_WRITE_FIELD_ID

Con esta macro se indica que quiere recuperar el identificador del campo de escritura del elemento.

M4_ITEM_READ_OBJECT_ID

Con esta macro se indica que quiere recuperar el identificador del objeto de lectura del elemento.

M4_ITEM_WRITE_OBJECT_ID

Con esta macro se indica que quiere recuperar el identificador del objeto de escritura del elemento.

M4_ITEM_SLICE_TOTALIZE

Con esta macro se indica que quiere recuperar el tipo de totalización que tiene el elemento para cuando se totalizan sus tramos.

M4_ITEM_NAME

Con esta macro se indica que quiere recuperar el nombre del elemento.

M4_ITEM_ORDER

Con esta macro se indica que quiere recuperar el orden del elemento dentro del nodo.

M4_ITEM_SYNONYM

Con esta macro se indica que quiere recuperar el sinónimo del elemento.

M4_ITEM_ID

Con esta macro se indica que quiere recuperar el identificadro del elemento.

M4_ITEM_NUMBER

Con esta macro se indica que quiere recuperar el número de elementos del nodo.

M4_ITEM_IS_PK

Con esta macro se indica que quiere recuperar si un elemento es PK del nodo o no. 1 si es pk y 0 si no es pk.

Ejecución de procesos con o sin tramos

Durante el periodo de ejecución de un proceso, la fórmula de un concepto o la de un método puede haber cambiado. También puede suceder que cambie el valor de alguno de los datos con los que trabaja el proceso de cálculo, por ejemplo, en el caso del cálculo de la nómina, puede cambiar la categoría laboral de un empleado, o el departamento al que está adscrito.

Se definen como ***elementos con tramos*** aquéllos cuyo valor o fórmula puede cambiar a lo largo del tiempo. Cada tramo del elemento está asociado a un periodo de validez que queda delimitado por una fecha de inicio y una fecha de fin.

Si se ejecuta el proceso sin tramos, éste calcula el valor de los distintos conceptos y métodos utilizando las fórmulas o reglas entre cuyas fechas de inicio y de fin esté comprendida la fecha de ejecución. De esta forma, para cada concepto o método, sólo se ejecuta una fórmula.

LN4 ofrece una serie de funciones que permiten acceder a las fechas de inicio y de fin del periodo, así como a la fecha de ejecución. Aunque estas fechas siempre se establecerán desde la interfaz del programa, LN4 incluye funciones para poder cambiar estas fechas durante la ejecución de un proceso.

Para obtener más información relativa a estas funciones, consulte el manual *Elementos tramados*.

Funciones de grabación en base de datos

En este apartado, se explican aquellas funciones que permiten realizar cambios de un modo u otro sobre la base de datos.

Existen dos tipos, su diferencia estriba en el ámbito de aplicación, ya que uno se produce a nivel de bloque (Blk) y otro, más general, se ejecuta en forma de cascada (Prg):

- **Blk**

  Las operaciones se realizan sobre un bloque de registros. Las funciones recogidas en este tipo son:

  – Delete_Blk
  – Update_Blk
  – Insert_Blk

- **Prg**

  Las operaciones se realizan sobre unos registros marcados y todos los que dependen de ellos. Las funciones recogidas en este tipo son:

  – Delete_Prg
  – Update_Prg
  – Insert_Prg.

Funciones de acción sobre un bloque

Delete_Blk ( )

Elimina de la base de datos todos los registros que hayan sido marcados como "borrados".

Esta función sólo elimina los registros que se hayan borrado en el bloque activo del nodo desde el que se llama a la función.

Por ejemplo, un modo de uso de esta función puede ser el siguiente:

- Inicie una transacción en la base de datos física.
- Ejecute la función *Detele_Blk ( )*.
- Finalice la transacción grabando permanentemente los cambios en la base de datos.

EJEMPLO

```
Begin Transaction
   Delete_blk ( )
End Transaction (Commit)
```

Update_Blk ( )

Modifica en la base de datos todos los registros que hayan sufrido cambios.

Esta función sólo actualiza aquellos registros que se hayan modificados en el bloque activo del nodo desde el que se llama a la función.

Por ejemplo, un modo de uso de esta función puede ser el siguiente:

- Inicie una transacción en la base de datos física.
- Ejecute la función *Update_Blk ( )*
- Finalice la transacción grabando permanentemente los cambios en la base de datos.

EJEMPLO

```
Begin Transaction
   Update_Blk ( )
End Transaction (Commit)
```

Insert_Blk ( )

Inserta nuevos registros en la base de datos.

Esta función sólo inserta aquellos registros que se hayan creado en el bloque activo del nodo desde el que se llama a la función.

Por ejemplo, un modo de uso de esta función puede ser el siguiente:

- Inicie una transacción en la base de datos física.
- Ejecute la función *InsertBlk ( )*.
- Finalice la transacción grabando los cambios permanentemente en la base de datos.

EJEMPLO

```
Begin Transaction
   Insert_Blk ( )
End Transaction (Commit)
```

Funciones de acción en cascada

Delete_Prg ( )

Elimina de la base de datos todos los registros que hayan sido marcados como "borrados", así como aquellos registros que dependan de ellos.

Esta función elimina los registros que se hayan borrado en el bloque activo del nodo desde el que se llama a la función y a todos sus hijos.

Por ejemplo, un modo de uso de esta función puede ser el siguiente:

- Inicie una transacción en la base de datos física.
- Ejecute la función *Detele_Prg ( )*.
- Finalice la transacción grabando los cambios permanentemente en la base de datos.

EJEMPLO

```
Begin Transaction
   Delete_Prg ( )
End Transaction (Commit)
```

Update_Prg ( )

Modifica en la base de datos todos los registros que hayan tenido cambios, así como los registros que dependan de ellos.

Esta función sólo actualiza los registros que se hayan modificados en el bloque activo del nodo desde el que se llama a la función y todos sus hijos.

Por ejemplo, un modo de uso de esta función puede ser el siguiente:

- Inicie una transacción en la base de datos física.

- Ejecute la función *UpdatePrg ( )*
- Finalice la transacción grabando los cambios permanentemente en la base de datos.

EJEMPLO

```
Begin Transaction
   Update_Prg ( )
End Transaction (Commit)
```

Insert_Prg ( )

Inserta nuevos registros en la base de datos.

Esta función sólo inserta aquellos registros que hayan sido creados en el bloque activo del nodo desde el que se llama a la función y todos sus hijos.

Por ejemplo, un modo de uso de esta función puede ser el siguiente:

- Inicie una transacción en la base de datos física.
- Ejecute la función *Insert_Prg (*).
- Finalice la transacción grabando los cambios permanentemente en la base de datos.

EJEMPLO

```
Begin Transaction
   Insert_Prg ( )
End Transaction (Commit)
```

Persist_Tree ( )

Graba en la base de datos física los cambios que se hayan efectuado en el bloque activo del nodo sobre el que se está posicionado, y en los bloques de los nodos hijo con los que está conectado.

La función *Persist_Tree ( )* ejecuta las siguientes acciones:

- Inicia una transacción en la base de datos física. Para ello, llama a la función *Begin Transaction ( )*.

- Posteriormente, realiza una de las siguientes acciones, dependiendo de la operación que se haya realizado previamente.
  - Borra de la base de datos física aquellos registros que se hayan marcado como borrados, bien desde la interfaz, bien mediante llamadas a la función *DeleteRegister ( )*.
  - Actualiza en la base de datos física los registros que se hayan actualizado, bien desde el interfaz del programa, o bien mediante llamadas a la función *UpdateRegister ( )*.
  - Inserta en la base de datos física los registros que se hayan añadido desde memoria, bien desde la interfaz del programa, o bien mediante llamadas a la función *InsertRegister (*).
- Cierra la transacción y puede realizar una de las dos opciones posibles:
  - Grabar permanentemente todos los cambios si éstos se han realizado con éxito (*Commit*).
  - Descartar los cambios, si alguno de los pasos anteriores no se pudo realizar satisfactoriamente (*Rollback*).